MASTER EN SCIENCES INFORMATIQUES

Outil d'aide à la sécurité et à l'administration d'un système UNIX

Alexandre, Simon

Award date:
2001

Simon ALEXANDRE

Année académique
2000-2001

MEMOIRE DE LICENCE A

Outil d'aide à la sécurité et à l'administration d'un système UNIX

Promoteur : Mr Radu V. COTET

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE

Au seuil de ce travail, je tiens à adresser tout particulièrement mes remerciements à Monsieur Radu Cotet qui, tout au long de mes recherches, m'assura constamment son entière disponibilité et ses précieux conseils.
Je remercie également Monsieur Diego Zamboni, auteur d'AAFID, pour l'éclairage qu'il a apporté à mes nombreuses questions.
Enfin, je voudrais aussi remercier toutes les autres personnes qui, d'une manière ou d'une autre, ont permis que ce mémoire voie le jour.

OUTIL D'AIDE A LA SECURITE ET A L'ADMINISTRATION D'UN SYSTEME UNIX
RESUME / ABSTRACT

Ce mémoire part de la constatation suivante : la tâche qui consiste à surveiller un système UNIX devient de plus en plus difficile étant donné le nombre sans cesse croissant de nouveaux utilisateurs. Par conséquent, la solution à ce problème se trouve dans l'automatisation de certains aspects de cette surveillance. Pour ce faire, nous avons recherché des outils (logiciels ou simples scripts) en langage Perl susceptibles d'intégrer ce type de routines. Cette recherche nous a amené à la découverte d'AAFID (Autonomous Agent For Intrusion Detection) un logiciel gratuit entièrement implémenté en Perl 5 et consacré à la surveillance d'un système UNIX. Cet IDS (Intrusion Detection System) utilise des agents qui sont chargés d'inspecter un aspect particulier du système. Sur cette base, notre travail a consisté dans un premier temps à installer AAFID et nous familiariser à son mode de fonctionnement. L'étape suivante s'est articulée autour de l'intégration de nouveaux agents à AAFID chargés d'effectuer certaines tâches de sécurité et d'administration nécessaires au système UNIX de l'Institut d'informatique. Ces agents ont été spécifiés en tenant compte du contexte dans lequel ils allaient évoluer. Ils ont ensuite été implémentés et testés sur les serveurs de l'Institut.

This dissertation starts from the following observation: the task which consists in supervising a UNIX system unceasingly becomes difficult being given the number crescent of new users. Consequently, the solution to this problem is the automation of certain aspects of this monitoring. With this intention, we sought tools (software or simple scripts) in Perl language able to include this type of routines. This search brought us to discovere AAFID (Autonomous Agent For Intrusion Detection), a free software entirely implemented in Perl 5 and dedicated to the UNIX system monitoring. This IDS (Intrusion System Detection) uses agents which are in charge to inspect a particular aspect of the system. On this basis, our work initially consisted to install AAFID and to familiarize us with his operating mode. The next step consisted in the integration of new agents in AAFID in order to realize certain tasks of security and administration necessary to the UNIX system of the Institute of computer Sciences. These agents were specified by taking into account the context in which they are going to evolve. They then were implemented and tested on the servers of the Institute of computer Sciences.

TABLE DES MATIERES

INTRODUCTION

1. Introduction générale

L'actuel accroissement de tous les parcs informatiques au sein des diverses institutions rend de plus en plus difficile la tâche d'administration d'un système. En effet, lorsqu'un administrateur était chargé de gérer quelques machines utilisées par plusieurs dizaines d'utilisateurs peu de problèmes se posaient alors. Le contexte actuel cependant conduit les administrateurs systèmes à se doter d'outils capables de les seconder, grâce à une certaine automatisation, dans leur travail de gestion des différentes parties du système.

C'est dans cette optique que s'inscrit le présent mémoire. Notre démarche a consisté, dans un premier temps, à rechercher des outils, principalement sous la forme de scripts en Perl, capables d'effectuer certaines routines d'administration et de surveillance de la sécurité. Cette recherche nous a permis de découvrir AAFID2[1], un logiciel de détection d'attaque sous UNIX, entièrement implémenté en Perl par des chercheurs de l'université de Purdue aux Etats-Unis. Le mode de fonctionnement de cet outil et la capacité, au sens technique et légal, d'y intégrer de nouvelles fonctionnalités nous ont immédiatement séduits. Nous pouvions envisager d'utiliser AAFID2 comme outil de surveillance et de gestion de notre système. La suite du travail a consisté à définir précisément les tâches d'administration ou de sécurité nécessaires à l'institut d'informatique que nous voulions implémenter pour pouvoir les intégrer dans AAFID2.

AAFID2 tournant sous UNIX, esquissons, avant d'entrer dans le vif du sujet, les traits principaux du système d'exploitation UNIX et des diverses tâches qui incombent à son administrateur et, plus particulièrement, celles qui relèvent du volet sécurité du système.

Pour commencer, un petit historique s'impose afin de bien situer ce système d'exploitation très prisé qu'est UNIX. Son histoire commence dans les années 60 aux Etats Unis lorsque AT&T, General Electric et le Massachusetts Institute of Technology s'allient pour réaliser un énorme projet : MULTICS (Multiplexed Information and Computing Service).[GAR 96 : 8-9] Cependant le projet trop ambitieux prend un retard considérable si bien que certains partenaires décident de l'abandonner. Ken Thompson, un chercheur d'AT&T continue alors les travaux en limitant les ambitions initiales du projet. [GAR 96 :9] En 1969 la première version du système voit le jour et est nommée UNIX.

[1] AAFID est un acronyme qui signifie Autonomous Agent For Intrusion Detection. Nous reviendrons plus en détail sur cet IDS.

Les années suivantes Thompson et ses collaborateurs continuent de travailler sur le système en y ajoutant de nouvelles fonctionnalités. En 1973, ils le reprogramment en langage C. [GAR 96 : 10] Dans le même temps, l'université californienne de Berkeley achète le code source complet du système UNIX d'AT&T et décide de continuer son amélioration dans ses propres laboratoires. C'est ainsi que l'on voit apparaître, dès 1977, à côté des versions V1, V6 et V7 d'AT&T les nouvelles BSD 1.0[2], BSD 2.0 de l'université de Berkeley. [PEL 96 : 29] Depuis ce moment là, le système UNIX ne cessa d'être amélioré parallèlement par plusieurs sociétés ou universités pour donner naissance à de nouvelles implémentations de ce système d'exploitation, parmi lesquelles Linux ou Solaris. [GAR 96 :14]

[2] BSD signifie Berkeley Software Distribution.

1.1. Caractéristiques générales du système d'exploitation UNIX

Nous allons à présent passer en revue les caractéristiques essentielles de ce système en soulignant celles dont nous reparlerons plus en détails lorsque nous aborderons la sécurité du système dans le chapitre suivant.

Unix a été conçu dès le départ pour être un système d'exploitation multi-utilisateurs et multi-tâches. Ces deux caractéristiques ont influencé la manière dont le système est construit et fonctionne. Examinons donc les implications majeures de ces deux choix.

a. Le système de fichiers

Les concepteurs d'UNIX ont opté pour une organisation hiérarchique des fichiers qui facilite ainsi leur organisation et leur gestion. On distingue donc les fichiers normaux (textes, exécutables ou binaires,...), les répertoires et les fichiers spéciaux (devices).

Le point de départ de ce système de fichier est la racine (notée /) sur laquelle viennent se greffer les autres répertoires. Le système comprend un ensemble de répertoires de base auquel viennent s'ajouter les répertoires créés pour et par les utilisateurs du système. Les répertoires de base sont : *home, bin, lib, boot, usr, dev, etc, tmp ,vmunix.* [PEL 96 :46] Ces répertoires contiennent des fichiers. Chaque fichier possède trois permissions qu'on peut appliquer aux trois classes d'utilisateurs : le propriétaire (*owner*), le groupe (*group*) et les autres utilisateurs du système (*others*). [PEL 96 :46] Les trois permissions sur les fichiers sont l'écriture (w *write*), la lecture (r *read*) et l'exécution (x *executable*). [WIE 00 : 105] Les mêmes permissions s'appliquent aux répertoires, excepté la dernière toujours caractérisée par *x* qui signifie ici non pas l'exécution mais le droit de traverser ce répertoire. [WIE 00 : 105]

b. Les utilisateurs

Le système distingue d'une part les utilisateurs et d'autre part les groupes qui se composent de plusieurs utilisateurs. Pour accéder au système, un utilisateur doit préalablement être enregistré par l'administrateur qui détient tous les droits d'accès au système. Il est désigné par le terme *root* ou super-utilisateur.[WIE 00 :321]

L'administrateur enregistre donc les nouveaux utilisateurs dans le système, crée les groupes lorsque c'est nécessaire. Nous reviendrons plus en détail sur le travail de l'administrateur.

c. Les réseaux

A l'heure actuelle, la grande majorité des machines, au sein d'une institution, sont installées en réseau afin de faciliter les échange de données entre elles et avec l'extérieur grâce à une connexion à l'Internet ou à un autre réseau.

Plusieurs protocoles sont mis en œuvre dans les systèmes UNIX. Le modèle de référence sous UNIX reste TCP/IP (*Transfert Control Protocol /Internet Protocol*) qui est utilisé à plusieurs niveaux. Il s'agit tout d'abord d'un moyen très répandu d'échange de données entre machines au niveau de la couche transport et réseau. [TAN 99 : 541-544] La notion de TCP/IP regroupe également plusieurs programmes dont le fonctionnement est basé sur ces protocoles réseau.[WIE 00 :391]

Les protocoles d'application utilisés sous UNIX sont les suivants : Les plus connus sont FTP (*File Transfer Protocol*), LPD (*Line Printer Daemon*), NFS (*Network File System*) , SMTP (*Simple Mail Transfer Protocol*), TELNET (*Terminal NETwork protocol*).[WIE 00 : 398-399] Nous rencontrerons certains d'entre eux dans le chapitre suivant lorsque nous traiterons de la sécurité du système.

1.2. Les tâches de l'administrateur système

Un système d'exploitation comme UNIX, lorsqu'il tourne en réseau, nécessite qu'une personne s'en occupe à tout moment. Cette personne c'est l'administrateur système. Nous allons examiner les diverses tâches que ce dernier doit effectuer et nous nous attarderons sur celles qui ont trait à la sécurité et à l'administration, objets de ce travail. On peut dénombrer neuf grandes tâches qui incombent à l'administrateur.

C'est lui qui va ajouter et supprimer les nouveaux utilisateurs, du nouveau matériel, effectuer les sauvegardes, installer de nouveaux logiciels, surveiller le système, s'occuper des pannes, assurer un minimum de documentation sur le système local, gérer la sécurité et, enfin, aider les utilisateurs. [NEM 95 : 9-11]

Parmi ces neuf tâches, deux nous intéressent particulièrement : la surveillance du système (*monitoring*) et la gestion de la sécurité.

a. La surveillance du système (*monitoring*)

La surveillance du système part de la constatation suivante : dans un réseau chaque utilisateur travaille sur sa machine et peut accéder aux ressources disponibles sur d'autres machines (stations de travail, imprimantes ou serveurs). Ces accès aux ressources reposent sur des services réseaux qui doivent être opérationnels à tout moment. L'administrateur va donc devoir vérifier qu'il n'y a pas de panne sur le réseau et que toutes les ressources sont disponibles. Pour ce faire, il dispose d'outils logiciels implémentés sous UNIX. Cependant il est inconcevable que l'administrateur passe son temps à vérifier en permanence l'état du système. C'est pourquoi il existe les fichiers *logs* dans lesquels sont enregistrées les données relatives à l'exécution de certaines applications, l'utilisation de ressources particulières. [NEM 95 :200-217] Ces fichiers constituent les archives du système, une sorte d'audit quotidien. L'analyse de ces fichiers permet d'une part de percevoir les dysfonctionnements du système et, d'autre part, se révèle très utile après une attaque.

b. La gestion de la sécurité

La sécurité d'un système constitue, comme nous allons le voir, un enjeu énorme au sein de chaque institution. Cette mission incombe le plus souvent à l'administrateur système qui se trouve face à plusieurs solutions pour gérer la sécurité de son système. Il peut vérifier l'intégrité de son système à l'aide de commandes déjà implémentée dont il analyse le résultat de l'exécution sous forme de fichiers de logs. Il peut également installer un logiciel de détection des attaques pour surveiller l'état de son système.

Au sein d'un système UNIX comme Solaris, on trouve un ensemble de fichiers logs de base. Ces fichiers sont stockés, à quelques exceptions près, des les répertoires suivants : */var/adm* ou */var/log/syslog* [GAR 96 : 291]. Ces répertoires contiennent les fichiers de logs à proprement parler. Par exemple, loginlog qui enregistre toutes les tentatives de login échouées. Ou encore *syslog* qui est un fichier enregistrant tous les messages générés par syslog. Ensuite, le fichier sulog, très important, car il enregistre toutes les utilisations de la commande SU. Enfin, le fichier xferlog dans lequel toutes les opérations FTP sont inscrites [GAR 96 : 303].

CHAPITRE I DETECTION DES INTRUSIONS. PRESENTATION ET ETAT DE LA QUESTION.

1.1. Introduction

Le géant de l'Internet Yahoo a été victime lundi d'une attaque « masquée » de grande envergure qui a bloqué son service pendant près de trois heures (...). [SOIR 00-a]

Mardi, une nouvelle salve a virtuellement fermé les boutiques du libraire Amazon.com, du géant de la vente aux enchères eBay et de buy.com, numéro deux de la vente aux particuliers sur Internet. [SOIR 00-b]

7 October 1999: *Hackers apparently working from Russia have systematically broken into Defense Department computers for more than a year and took vast amounts of unclassified but nonetheless sensitive information, U.S. officials said Wednesday. Besides penetrating the Pentagon's defenses, the hackers have raided unclassified computer networks at Energy Department nuclear weapons and research labs, at the National Aeronautics and Space Administration and at many university research facilities and defense contractors, officials said.*[ALL 00-a: 1]

Septembre 1998: *Hackers are banding together across the globe to mount low-visibility attacks in an effort to sneak under the radar of security specialists and intrusion detection software, a U.S. Navy network security team said today. Coordinated attacks from up to 15 different locations on several continents have been detected, and Navy experts believe that the attackers garner information by probing Navy Web sites and then share it among themselves. "These new patterns are really hard to decipher — you need expert forensics to get the smoking gun," said Stephen Northcutt, head of the Shadow intrusion detection team at the Naval Surface Warfare Center. "To know what's really happening will require law enforcement to get hold of the hackers' code so we can disassemble it."*.[ALL 00-a : 3]

Voici quatre « faits divers » informatiques de ces dernières années liés au célèbre « piratage informatique » dont on parle de plus en plus fréquemment aujourd'hui. Œuvre de hackers[3] le plus souvent anonymes, ces attaques préoccupent hautement les ténors de la nouvelle économie et les responsables de réseaux informatiques. Les premières attaques recensées remontent au milieu des années 80. A l'époque, elle sont l'œuvre de professionnels de l'informatique qui mettent au point des techniques d'attaque à partir de leur ordinateur personnel. Les statistiques du CERT/CC[4] sont claires : en 1989 une centaine d'incidents sont enregistrés. En 1999, près de 8400. [ALL 00-a : 4] En outre, avec l'explosion de l'Internet, les outils de base nécessaires pour attaquer un ordinateur circulent on ne peut plus aisément.

---

[3] A l'origine le terme *hacker* n'était pas péjoratif. La définition citée par Garfinkel et Spafford [GAR 96 : 4] le montre bien : *1. A person who enjoys learning the details of computer systems and how to stretch their capabilities – as oppsosed to most users of computers, who prefer to learn the minimum amount necessary. 2. One who programs enthusiastically or who enjoys programming rather than just theorizing about programming.* Les auteurs précisent toutefois qu'actuellement le terme de hacker désigne les pirates informatiques et plus les passionnés. On trouvera également dans la littérature sur le sujet le terme de cracker. [GAR 96 : 4]

[4] Le CERT/CC (Computer Emergency Response Team) est un centre de recherche en matière de sécurité informatique. Créé en 1988 aux Etats-Unis après l'attaque du virus « Morris » qui a affecté près de 10% des ordinateurs connectés à l'Internet. http://www.cert.org

La sécurité informatique est donc un problème de taille à l'heure du commerce électronique et de l'extension des réseaux. Le problème est clairement posé à l'occasion d'une table ronde organisée par le CERIAS[5] : *Today, we make sure that the doors of our houses are securely locked.*(...) *but are all our virtual doors secure ?.* [CER 00 : 9] L'ampleur du problème est considérable à en croire la masse d'informations disponible sur le sujet. Les centre de recherche privés, publics et universitaires ne cessent de traquer les failles connues dans les systèmes afin de les combler en vue d'assurer une sécurité maximale des réseaux informatiques et des données auxquelles ils donnent accès.

Jusqu'à présent, nous avons parlé de sécurité informatique, d'incidents ou d'attaques, de hackers. Termes qu'il s'agit de préciser afin d'éviter toute confusion.

[5] CERIAS. Center for Education and Research in Information Assurance and Security. At Purdue University. http://www.cerias.purdue.edu/

1.2. Concepts de sécurité informatique

1.2.1. Sécurité informatique : terminologie générale.

Précisons, dans un premier temps, les termes généraux que nous allons employer tout au long de ce travail. L'abondante littérature sur le sujet utilise essentiellement deux termes : *attack* et *intrusion.* Dans un ouvrage récent, plusieurs spécialistes de la question proposent une définition de ces termes finalement assez proches sémantiquement.[ALL 00-a] Ils définissent *attack* en ces termes :*An action conducted by one adversary, the intruder, against another adversary, the victim. The intruder carries out an attack with a specific objective in mind. From the perspective of an administrator reszponsible for maintaining a system, an attack is a set of one or more events that may have one or more security consequences. From the perspective of an intruder, an attack, is a mechanism to fulfill an objective.* [ALL 00-a : 8] La définition proposée pour *intrusion* ressemble d'assez près à la précédente. *Intrusion : A common synonym for the word « attack » ; more precisely, a successful attack.* [ALL 00-a : 8] Nous allons à présent passer en revue les notions communes aux divers types d'attaques. Il ne s'agit pas d'une énumération exhaustive de types d'attaques ni de moyens mais plutôt d'un essai de typologie. [HOW 98]

Tout d'abord, il s'agit de distinguer la notion d'événement comme un changement discret d'état ou de statut d'un système ou un périphérique. [HOW 98 : 6] Ces changements d'état sont la conséquence d'actions, au sens large, visant des cibles spécifiques. Nous retiendrons les définitions suivantes du terme action . *Probe* : obtenir les caractéristiques d'une cible ; *Scan* : passer en revue les caractéristiques de plusieurs cibles ; *Flood* : accéder à maintes reprises à une cible afin de la saturer ; *Authenticate* : usurper une identité ; *Bypass :* éviter une identification obligatoire ; *Spoof :* déguiser son identité ; *read, copy, steal, modifiy* et *delete* dont le sens n'est plus à préciser.

On passe dans l'illégalité à partir du moment où la source ne possédant aucun droit sur la destination tente d'accéder aux ressources de cette dernière. Les événements qui ont alors lieu sur un ordinateur ou un réseau constituent les étapes successives d'un processus totalement illégal. Ces événements font dès lors parties d'une attaque au sens strict. [HOW 98 : 11]. Ces attaques visent toujours une cible bien particulière qui peut être de type différent : *account* :compte d'un utilisateur sur une machine ; *process :* un programme en cours d'exécution *; data :* des données ; *computer* et/ou *component , network* ou *internetwork.* [HOW 98 : 11] L'auteur de l'attaque va, bien entendu, profiter d'une faiblesse connue de la cible . On distingue trois types de faille : *design* (due à la conception ou spécification logicielle et/ou matérielle), *implementation* ou *configuration vulnerability.* [HOW 98 : 14]

Dernier élément nécessaire à l'attaque : l'outil. Par outil il faut entendre tout moyen utilisé pour exploiter une faille dans un ordinateur ou un réseau. Cela va de la simple commande au programme d'attaque élaboré [6]. Leur nombre sans cesse croissant rend impossible toute énumération mais l'on peut toutefois établir certaines catégories d'outils utilisés pour attaquer un ordinateur ou un réseau. [HOW 98 : 13]

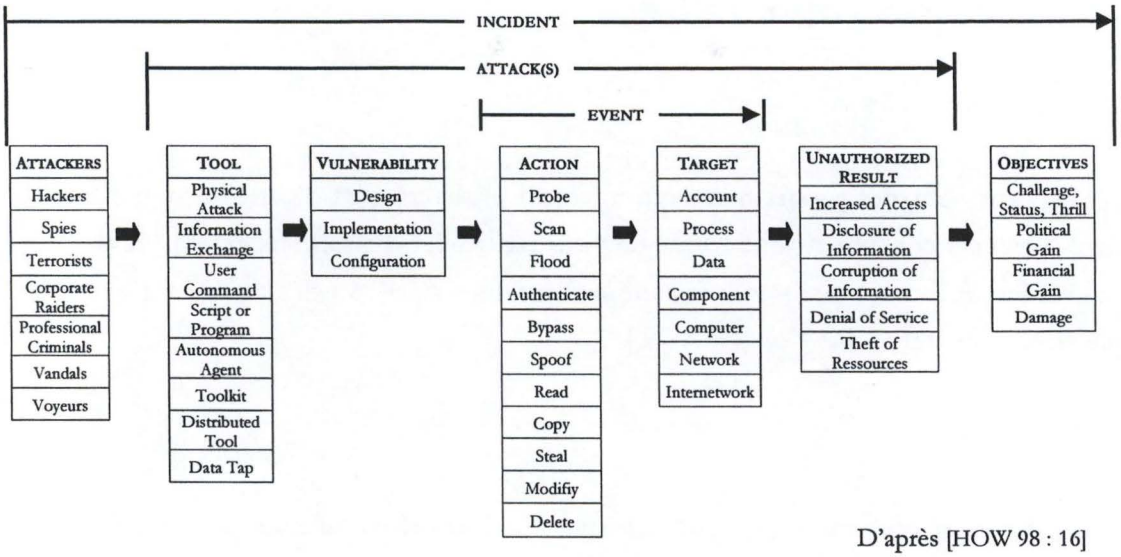
- *user command :* exploiter les failles d'un système en utilisant les commandes de programmes pré-installés tels que Telnet, Ping.
- *script* ou *programme* : consiste en l'exécution d'un programme ou d'un fichier de commandes (script) dont les instructions participent à la réussite d'une attaque.
- *autonomous agent :* programme ou fragment de programme qui travaille indépendamment de l'utilisateur. Par exemple, les virus ou les vers.
- *toolkit :* un package de programmes contenant des scripts, programmes ou agents autonomes qui exploiteront les failles du système.
- *distributed tool :* outil pouvant être distribué sur plusieurs hôtes qui seront coordonnés afin d'accomplir simultanément une attaque à un moment donné.

Notons que l'Internet est devenu, à ce sujet, le plus grand vecteur d'échange d'informations et d'outils utilisés lors des attaques de réseaux informatiques.

En cas d'attaque réussie, on se trouve face à plusieurs types de résultats possibles : [HOW 98 : 14]

- *increased access* : une voie d'accès non autorisée est ouverte dans un système.
- *diclosure of information* : de l'information à caractère privé est volée.
- *corruption of information* : modification de données stockées par la cible.
- *denial of service* : dégradation ou blocage des ressources de la cible.

[6] Par exemple un virus ou encore un cheval de Troie (Trojan horse). Voir définition ci-dessous.



D'après [HOW 98 : 16]

Voici, résumés sous forme de tableau, les différents éléments qui participent à ce que l'on appelle une attaque, une intrusion ou encore *Security Incident* . Voyons à présent les types d'attaques les plus fréquemment rencontrés. On retrouvera à cette occasion, seuls ou combinés, plusieurs types d'action évoqués ci-dessus.

1.2.2. Les principaux types d'attaque.

a. Les virus et les vers (worms)

Les virus sont des programmes qui infectent et détériorent d'autres programmes en les modifiant. Les vers constituent une forme de virus particulier. Ils se transmettent de machine en machine via les réseaux (e-mail) en infectant les machines sur lesquelles ils ont été exécutés. Un exemple célèbre de ver : *I love You.* [NSA 99]

b. Sniffers

Un sniffer se présente le plus souvent sous une forme logicielle. Le principe du sniffer est relativement simple : collecter toutes les informations passant sur le réseau à l'endroit où il est positionné. [NSA 99] L'intérêt d'un tel programme réside dans la capture d'informations telles que les login et mot de passe d'un utilisateur ou administrateur ou tout autre type de données. [TAB 98]

c. IP Spoofing

La traduction littérale d'IP Spoofing signifie "parodie d'IP" et il s'agit bien de cela. L'IP Spoofing est une technique découverte dès 1985 qui consiste à déguiser la véritable identité que constitue l'adresse IP de l'ordinateur que l'on utilise en la modifiant. [TAB 98] Ce type de technique est de plus en plus utilisé afin d'éviter d'éventuelles poursuites de la part des victimes qui sont incapables de connaître la véritable identité des auteurs de l'attaque.

d. Denial-of-Service (DOS) et Distributed Denial-of-Service (DDOS)

Le "déni de service" représente un des types d'attaque les plus répandus aujourd'hui. Ces attaquent provoquent de grosses pertes financières et opérationnelles chez la victime . La plupart du temps, les cibles sont des sites web de commerce électronique, d'entreprise ou d'universités. [DIS 00] Pendant et après une attaque DOS ou DDOS, un système pris pour cible est incapable durant un certain laps de temps (la remise en état du système) de répondre aux demandes de ses clients.

Une attaque DOS a pour objectif de saturer, depuis une machine extérieure, les ressources du système pris pour cible en le noyant d'informations sur un laps de temps très court. Les moyens généralement utilisés sont les suivants [DIS 00] :

- *SYN flood :* une série de paquets SYN (ouverture de connexion TCP) est envoyée ouvrant ainsi un nombre sans cesse croissant de connexions.

- *UDP flood :* des paquets UDP sont envoyés en masse sur tous les ports d'une machine. Celle-ci tente de répondre par des messages d'erreur (ICMP) augmentant de la sorte le flux de messages participant à la saturation du réseau.
- *ICMP flood :* utilisation en masse de requête PING vers une même machine jusqu'à la saturation du système.

Les attaques DDOS, dites distribuées, fonctionnent exactement sur le même mode à la différence qu'il n'y a plus une source d'attaque mais plusieurs. En général, les auteurs de l'attaque se servent de serveurs intermédiaires mal protégés d'où ils lancent, au même moment, une attaque vers une machine cible.[RES 99 : 5-7] L'attaque lancée en février 2000 sur Yahoo !, Amazon.com et autres fait partie de ce genre d'attaques. Au plus fort de l'attaque de Yahoo !, le flux de demandes adressées à Yahoo ! a atteint le volume de un milliard de bits par seconde ce qui représente le volume de demandes atteint par certains sites sur un an. [SOIR 00-a]

e. Trojan Horse (cheval de Troie)

Un cheval de Troie (informatique) est un programme caché dans un autre qui exécute des commandes sournoises. Un peu comme le virus, le cheval de Troie est un code (programme) nuisible placé dans un programme sain (imaginez une fausse commande *dir*, qui normalement liste les fichiers, qui vous détruit les fichiers au-lieu d'en afficher la liste). Il exécute des instructions nuisibles lorsque vous exécutez le programme sain. Il peut par exemple voler des mots de passe, copier des données, ou exécuter tout autre action nuisible ... Un tel programme peut créer, depuis l'intérieur du réseau, une brèche volontaire dans la sécurité afin d'autoriser l'accès à des parties protégées du réseau. La détection d'un tel programme est difficile car il faut arriver à détecter les symptômes dus à l'action du programme (le cheval de Troie) . [TAB 98]

Les RootKit sont classés dans la catégorie des chevaux de Troie étant donné leur mode de fonctionnement. Ils sont utilisés par les hackers pour dissimuler et maintenir leur présence au sein d'un système qu'ils ont réussi à pénétrer.[BRU 99 : 27] La première étape consiste à entrer sans éveiller l'attention dans le système et s'y dissimuler. Ensuite, lorsqu'on a effacé toutes les traces de son passage, il faut ouvrir une porte (*backdoor*) dans le système afin d'y revenir ultérieurement. [BRU 99 : 27] Un RootKit contient tous les outils logiciels nécessaires à la réalisation de ces étapes. Une fois introduit dans le système, l'intrus va usurper les droits de l'administrateur sur les commandes les plus importantes. [BRU 99 : 28] Notons que seul un logiciel de détection pointu sera en mesure de détecter les programmes corrompus. L'administrateur seul ne pourra pas percevoir tous les programmes modifiés à l'aide des commandes traditionnelles. [BRU 99 : 28] Toutefois l'expérience montre que la grande majorité des hackers ne parvient pas à effacer entièrement les traces de leur passage rendant ainsi possible leur détection. [BRU 99 : 28-29]

1.2.3. La détection des intrusions.

La détection des intrusions, en tant que secteur de recherche particulier de la sécurité informatique, date des années 80.[FRI 00 : 541] Le peu de réseaux et d'interconnexions entre réseaux durant ces années, ne motive pas le développement de la recherche en matière d'IDS[7]. Par conséquent, la demande pour ce type de technologie étant relativement faible, la recherche n'est pas approfondie et les quelques systèmes disponibles sont peu performants. [FRI 00 : 541] Le développement du World Wide Web et la fréquence croissante des attaques informatiques, dans les années 90, a engendré le renforcement du secteur de recherche en matière de détection des intrusions. [FRI 00 : 542] [ALL 00-a : 7] Les IDS ont été développés parallèlement aux firewalls afin de protéger les réseaux locaux. Actuellement, les progrès de la recherche sur les réseaux, l'augmentation des interconnexions posent de nouveaux enjeux de taille aux IDS. En effet, la plupart des IDS développés par des sociétés privées présentent de nombreux défauts exploités par les hackers. [FRI 00 : 542] De plus, les attaques sont devenues de plus en plus complexes et difficiles à contrecarrer, sans parler des attaques distribuées (Ddos) dont la récente augmentation ne présage rien de bon étant donné le manque de moyens pour les contrer. Enfin, le nombre sans cesse croissant de nouveaux internautes, dont, parmi ceux-ci, des pirates potentiels, ne fait qu'augmenter les risques d'attaques. [FRI 00 : 542] Les conférences scientifiques sur le sujet, RAID'98[8] et RAID'99, posent les domaines de recherches essentiels : l'identification des pirates, les procédures légales, les systèmes automatiques. [FRI 00 : 543] La lutte contre les virus est une autre branche spécialisée de la sécurité informatique. L'augmentation de la fréquence des attaques a engendré la création au sein d'entreprises d'unités spécialisées dans la prévention et la réparation d'intrusions. On parle alors de CSIRT pour *Computer Security Incident Response Teams.* [WES 98]

a. Les *IDS* (*Intrusion Detection System*)

Les IDS sont des outils logiciels que l'on implante sur des machines susceptibles d'être attaquées.[9] Un IDS peut aussi bien être installé sur une machine isolée que sur une autre connectée à un réseau.[PUK 96 : 2] Le but d'un système de détection des intrusions consiste à analyser les informations afin d'identifier une véritable attaque tout en évitant les fausses alertes.[HUG 00] La détection des intrusions peut être vue comme un simple problème de détection de signaux. Les manifestations d'intrusion sont ici les signes à détecter. Un IDS va donc

---

[7] IDS : *Intrusion Detection System*

[8] RAID : *International Workshop on Recent Advances in Intrusion Detection (RAID)*

[9] N.B : il s'agit ici de d'introduire la notion de *firewall* également très souvent rencontrée dans la littérature consacrée à la sécurité des réseaux informatiques. A la différence d'un IDS, un *firewall* est un système que l'on installe entre deux ou plusieurs réseaux (le plus souvent à l'entrée d'un LAN). Le *firewall* va filtrer toutes les informations passant par lui sur le réseaux selon des modalités fixées par l'administrateur du réseau qui l'a configuré. [NSA 99]

devoir distinguer les signaux d'une activité normale des bruits provoqués par une attaque[10]. [HUG 00]

Pour détecter une intrusion, un IDS doit donc analyser en permanence le système qu'il protège. Nous verrons plus loin les secteurs critiques d'un système qui doivent être particulièrement surveillés par l'IDS. On peut trouver via l'Internet toute une série d'IDS proposés par des sociétés spécialisées dans la sécurité informatique. Lorsque l'IDS a été élaboré par une société privée, il est payant. Par contre, dans la plupart des cas, les IDS mis au point par les départements d'informatiques des universités sont proposés gratuitement.

Enfin, avant de passer à l'étude plus approfondie du fonctionnement des IDS, il ne faut pas perdre de vue, comme le souligne un rapport du CERT, que la première chose qu'un hacker va analyser en pénétrant le système est la présence ou non d'un IDS. Dans l'affirmative il s'emploiera dans un premier temps à tenter de le supprimer ou le rendre inefficace.[ALL 00-b]

b. Deux catégories de détection d'intrusions

Avant d'aborder les différentes catégories de systèmes existant, il est intéressant d'examiner les critères permettant d'évaluer l'efficacité d'un IDS. On peut distinguer cinq critères pertinents. [DEB 99 : 806-807] Le premier de ces critères est la précision du système qui ne prendra en compte que les signes anormaux révélant une véritable attaque. Ensuite vient la performance du système au point de vue rapidité de réaction, élément indispensable pour tout système de détection en temps réel. La complétude forme le troisième critère d'évaluation, pour sa part, assez théorique. Le caractère complet d'un IDS est purement théorique étant donné que cela suppose une parfaite et totale connaissance des types d'attaques potentiels, ce qui ressort de l'impossible. La résistance aux pannes peut également être vue comme un critère important. Ainsi que nous l'avons évoqué ci-dessus, l'IDS, pour être mis hors circuit, sera souvent la première cible de l'attaque. Cet élément doit donc être pris en compte lors de son implémentation. Le dernier critère à envisager est la rapidité, au sens large, de l'IDS. Il faut entendre ici la vitesse de détection de l'attaque et surtout de propagation de l'information dans le système pour que les responsables soit prévenus le plus tôt possible afin de limiter les dégâts.

On peut diviser la détection des intrusions en deux catégories *anomaly* et *misuse intrusion detection*[11].[KUM 95 : 6] [SUN 96 : 3] L'*anomaly intrusion detection* se réfère aux intrusions pouvant être détectées sur base d'un comportement anormal et de l'utilisation inhabituelle de ressources informatiques. Sur base des différents profils d'utilisateurs (horaires de connexion, utilisation des

[10] Les IDS vont combiner en général les deux techniques de détection des intrusions précédent évoquées.

[11] On trouve également d'autres termes synonymes pour qualifier ces deux catégories. On parlera pour *anomaly detection* de *detection by behavior* ou *behavior-based* et pour *misuse detection* de *detection by appearance* ou *knowledge-based.* [DEB 99 : 807]

ressources) au sein d'une institution on parvient à repérer une activité suspecte[12]. [SUN 96 :3] Effectivement, une personne extérieure qui tente de pénétrer le système ne connaîtra pas les usages du propriétaire de la machine qu'il utilise[13].[KUM 95 :7-8] Les avantages de ce système résident dans sa capacité à détecter les tentatives d'exploitation de nouvelles failles et donc les nouveaux types d'attaque.[DEB 99 : 809] Elles permettent également de percevoir les abus de privilège. Cependant l'*anomaly intrusion detection* est critiquée pour sa fâcheuse tendance à générer de fausses alertes. [DEB 99 : 809-810]

Quant à l'autre catégorie, *misuse intrusion detection*, elle se fonde sur des modèles prédéfinis d'attaques qui exploitent les faiblesses d'un système. De tels modèles pouvant être constitués à l'avance. [KUM 95 :6] A la différence de la technique précédente, il s'agit ici de détecter directement un comportement anormal sur la base de sa description précédemment élaborée. En d'autres mots, tout comportement qui n'est pas explicitement reconnu comme étant une attaque est considéré comme normal. La principale faiblesse de cette technique réside dans son mode de fonctionnement : elle ne lutte efficacement que contre les attaques qu'elle connaît.[KUM 95 :9] L'efficacité d'un tel système dépendra de ce fait de la quantité et la précision des informations qu'on lui a transmis. Ce choix d'implémentation implique également un travail permanent de maintenance du système. Chaque nouvelle faille découverte dans le système doit être « apprise » au système de détection afin de prévenir toute attaque éventuelle par ce biais.[DEB 99 :808-809]

En outre, un IDS peut être qualifié d'actif ou de passif selon son type de réaction en cas d'attaque. Par passif on entend un IDS qui génère simplement une alarme en cas de détection d'attaque.[DEB 99 : 811] Cette manière d'agir est implémentée dans la plupart des IDS disponibles. Par opposition, un IDS dit actif va déclencher une alarme et lancer un processus de lutte contre l'attaque. Notons cependant que ce dernier type d'IDS n'est pas fréquent car plus lourd à implémenter. Cela est dû, en grande partie, aux risques de fausses alarmes (voir point *g*) qui risquent de provoquer l'exécution d'un processus de réaction non approprié.

Enfin, on pourra distinguer les IDS selon leur lieu d'implantation : *host-based* ou *network-based.* [DEB 99 : 811] A l'heure de l'expansion croissante des réseaux, on parlera le plus souvent du second type d'IDS.

---

[12] A cette fin on utilise des statistiques tirées de l'exploitation habituelle du système. Pour cela on prend plusieurs variables que l'on mesure sur une période de temps donnée. Les données que l'on mesure sont en général le temps de connexion (login et logout time), l'utilisation des ressources, le pourcentage habituel d'utilisation processeur/mémoire. [DEB 99 : 810]

[13] L'auteur nuance toutefois ses propos en soulignant les failles de ce type de méthode. Il souligne que l'activité provoquée par une intrusion ne coïncide pas forcément avec une activité anormale. Il distingue quatre possibilités : *Intrusive but not anomalous, Not intrusive but anomalous, Not intrusive and not anomalous* et *Intrusive and anomalous.* [KUM 95 : 8]

c. Cycle de vie d'un IDS

Lorsque l'on parle d'IDS, il est important de considérer l'ensemble des étapes préalables à l'utilisation d'un tel dispositif. Les spécialistes distinguent traditionnellement quatre étapes : évaluation et sélection, déploiement, opérations et utilisation et maintenance. S'y ajoute parfois, comme étape préliminaire, l'installation d'une architecture sécurisée ou défense en profondeur. [ALL 00-a : 96]

Cette étape préliminaire apparaît comme essentielle avant l'installation de l'IDS. Son omission réduit considérablement l'efficacité future du système de détection des intrusions. En effet, il s'agit d'une série d'opérations dont l'objectif premier est de supprimer un certain nombre de failles présentes dans le système. Remarquons qu'elles peuvent être réalisées indépendamment de l'installation d'un IDS. Voici quelques éléments tirés d'une liste non exhaustive proposée par des spécialistes[14] : [ALL 00-a : 96]

- appliquer des patches aux applications fragiles, modifier certaines configurations du système
- supprimer les services superflus
- utiliser un ou plusieurs firewalls pour limiter l'accès
- implémenter des mécanismes d'authentification
- superviser régulièrement le réseau

Examinons à présent, une à une, les quatre étapes pré-citées.

1. Evaluation et sélection

Première étape, l'examen des systèmes disponibles afin de déterminer celui qui conviendra le mieux. L'évolution rapide des technologies, l'absence de normes définies, le flou quant à l'efficacité réelle des systèmes et la quantité de travail que leur utilisation entraîne, ne facilite pas ce choix. [ALL 00-b] Voici quelques critères à prendre en compte dans le choix d'un IDS : la précision du système (peu de fausses alarmes), la guidance en cas d'attaque, les performances (rapidité de détection), la solidité du logiciel, la facilité d'utilisation quotidienne, le coût éventuel. [ALL 00-a : 97-98] L'ensemble de ces éléments n'est pas à négliger lorsque l'on souhaite installer un IDS. Le risque encouru si l'on choisit mal un IDS réside dans le surplus de confiance accordé au système et à ses présumées performances. [LIP 00 : 580]

[14] On consultera également avec intérêt l'article [MEA 00] publié en septembre 2000 sous l'égide du SEI (Software Engineering Institute) intitlué *survivable Network Analysis Method* . Il expliquent les faiblesses et avantages des différentes configurations d'un réseau.

2. Déploiement

Une fois l'IDS choisi, il faut envisager les problèmes qui risquent de surgir, réfléchir à la politique que l'on va suivre et qui se traduira dans la configuration de l'IDS. Tout d'abord, établir la liste des endroits critiques du système où les capteurs seront installés. Ensuite, installer et configurer l'IDS en respectant les décisions de comportement prises auparavant. Enfin, établir, de manière précise, les procédures à suivre en cas d'attaque. Cela signifie concrètement, définir les divers profils d'attaque et leurs symptômes, identifier les procédures de collectes de données à analyser, ramasser toutes les preuves possibles. Tout cela dans le but de correctement évaluer une attaque et identifier ses auteurs. [ALL 00-a : 98-99]

3. Opérations et utilisation

Une fois installé, l'IDS doit être en permanence surveillé puisqu'en cas d'attaque il risque d'être pris pour cible en tout premier lieu. Examiner régulièrement les fichiers dans lesquels les diverses traces sont enregistrées par l'IDS. [ALL 00-b]

4. Maintenance

Comme beaucoup d'autres logiciels, les IDS sont susceptibles d'être remis à jour. Tout comme les anti-virus, ces mises à jour sont plus qu'indispensables étant donné l'évolution fulgurante des techniques d'attaques. De plus, il sera nécessaire d'introduire de nouveaux profils d'attaques ou de sécuriser certaines failles du système récemment découvertes. [ALL-a 00 :99]

d. Etablir un profil type du système à protéger

Revenons à présent plus en détail sur une des opérations essentielles à réaliser lors de l'implantation d'un IDS : l'établissement du profil « type » du système. Comme nous l'avons vu les IDS fonctionnent essentiellement sur deux grands modes : *anomaly* et *misuse*. La détection d'une anomalie suppose la confrontation de cette anomalie avec un modèle de référence préétabli. Par modèle de référence, il faut voir la collecte de données et l'établissement de la liste des comportements d'utilisation du système et de ses ressources .[CERT 00-a] Les divers types de données à collecter sont, par exemple, les performances habituelles du système et du réseau, la liste des répertoires et fichiers, des utilisateurs, des applications[15]. Plus le profil du système sera précis et documenté, plus il y aura de chances de détecter une éventuelle intrusion.

[15] Voir en fin de chapitre une liste des données à collecter proposée par le CERT.

e. Repérer les signes d'une intrusion

La présence d'un IDS n'empêche pas un hacker de pénétrer un système mais permet au moins de limiter les dégâts en détectant sa présence le plus rapidement possible. Une fois l'intrus détecté et mis hors du système, il est nécessaire de constater les modifications qu'il a pu éventuellement opérer sur le système[16]. Les mesures préconisées sont les suivantes : vérifier la présence de comptes suspects d'utilisateurs ou de groupes et, au sein de chaque groupe, vérifier la présence d'un nouvel utilisateur créé par l'intrus[17].[CERT 00-b] Ensuite, il faut d'une part inspecter les droits des utilisateurs sur les fichiers sensibles du système dans le cas où certains de ces droits auraient été modifiés et d'autre part vérifier qu'un processus étranger n'a pas été lancé. [CERT 00-b] Un passage en revue des fichiers du système avec un outil spécialisé permettra de découvrir si une ou plusieurs modifications ont eu lieu. En effet, après une attaque, un fichier peut avoir été modifié tout en conservant son apparence initiale. Seul un logiciel spécialisé permettra de détecter ce type de modifications[18]. [CERT 00-b] Il faudra également vérifier la présence d'éventuels sniffers placés sur le réseau, l'ajout éventuel de nouveaux services non autorisés. [CERT 00-b] Le fichier contenant les mots de passe, `/etc/shadow`, sera également à vérifier. Enfin, dernières vérifications à réaliser : la présence de fichiers cachés et une éventuelle modification de la configuration du système. [CERT 00-b]

f. Caractériser une intrusion

Nous venons, dans le point précédent, de passer en revue les différents éléments à travers lesquels il est possible de retrouver les traces d'une intrusion détectée ou non. Afin de correctement analyser l'objectif d'une intrusion, un peu de recul s'impose. Plusieurs questions posées cherchent encore une réponse que tous les indices, pris de manière isolée, ne peuvent apporter. Quel type d'attaque a été utilisé ? La réponse à cette question permettra de découvrir la faille dans le système, l'endroit où l'attaque a été portée.[KOS 99 : 25] Sur base des données manipulée par l'intrus, voir ce qu'il a fait après avoir obtenu l'accès au système. Enfin, il reste à tenter de savoir d'où provient l'attaque. A cette fin, l'examen des fichiers de logs des firewalls, routeurs et autres moniteurs de réseau s'avère souvent utile. [KOS 99 : 27] Les attaques laissent effectivement des traces qui permettent de remonter jusqu'au(x) système(s) utilisé(s) pour réaliser l'attaque principale.[Kos 99 :27-28]

---

[16] Un rapport du CIAC (*Computer Incident Advisory Capability*) datant de 1994 reprend les diverses commandes UNIX susceptibles d'êtres utilisées pour détecter les signes d'une intrusion. Malgré son ancienneté et l'évolution des techniques d'attaques les pistes soulevées dans ce rapport paraissent encore intéressantes. [PIC 94]

[17] L'utilisation de la commande *find* permet de repérer une anomalie dans la liste des utilisateurs et des groupes.

```
find / -user root -perm -4000 -print
find / -group kmem -perm -2000 -print
find / -user root -perm -4000 -print -xdev
```

[CERT 00-b]

[18] De tels outils comme MD5 ou Tripwire disponibles sur Internet analysent les checksum des programmes pour détecter les éventuelles modifications. Une liste de ces outils et de l'endroit où se les procurer est disponible sur les site du CERT. [CERT 00-c]

g. Problèmes des IDS : les fausses alarmes et le flux de données

Bien plus qu'une alarme traditionnelle, un IDS a la fâcheuse tendance à générer de fausses alertes. La cause réside dans le principe qui régit leur fonctionnement. Un IDS, comme nous l'avons vu, réagit aux informations générées par les divers capteurs placés au sein du système à protéger. Une alarme sera déclenchée si un capteur détecte un comportement inhabituel pour lui. Son déclenchement va générer, dans la plupart des systèmes, l'enregistrement d'un flux conséquent de données à analyser. Plus les capteurs son nombreux et sensibles, plus les fausses alarmes sont nombreuses et, par conséquent, plus la masse de données devient importante. [MAN 00 : 572] On peut distinguer deux causes à ces fausses alarmes. La première réside dans le manque d'informations fournies à l'IDS. Comme les capteurs de l'IDS fonctionnent par comparaison de comportements observés par rapport à des comportements connus, les imprécisions ou manques de descriptions sont sources d'erreurs. Ce constat réalisé par les utilisateurs d'IDS [MAN 00 : 571-573] tend à montrer l'importance de l'affinement des critères d'observation implémentés dans l'IDS. La seconde est souvent due à un mauvais choix d'IDS non approprié au système sur lequel il est implanté. La conséquence de ce mauvais choix se traduit par un nombre anormal de fausses alarmes qui génèrent un flot d'informations impossibles à analyser correctement. [LIP 00 : 580]

1.2.4. Scénario fréquent d'attaque.

Lutter contre les attaques sans en connaître et en comprendre le fonctionnement serait purement et simplement inutile et inefficace. Il s'agit de comprendre le mode de fonctionnement des attaques, les stratégies utilisées par les auteurs d'attaques afin de préparer du mieux possible les IDS. [HUA 99 : 2465] C'est pourquoi, après avoir présenté les systèmes de détection d'intrusions (IDS), nous allons à présent examiner un des scénarios potentiels d'attaque. Ce scénario est décrit dans un article de la revue *Computer Networks* consacré à l'analyse des stratégies d'attaque. [HUA 99 : 2465-2475] Il s'agit en l'occurrence d'attaques pensées, préparées et non pas de tentatives d'attaques perpétrées par des internautes en quête de frissons qui reprennent simplement et souvent maladroitement des outils trouvés sur le web.

Les étapes d'une attaque de réseau ne sont jamais réalisées dans un ordre aléatoire. L'auteur de l'attaque a à sa disposition tous les outils nécessaires pour assurer le succès de son entreprise. Ce dernier va, dans un premier temps, collecter le maximum d'informations possible sur le système cible. Le type et le nombre d'outils qui seront utilisés à cette fin dépendront surtout des informations récoltées et des réponses du système interrogé. Ensuite, l'étape suivante va consister à masquer son identité grâce à la technique de l'IP Spoofing. Une fois la connexion établie avec la cible, la suite de l'attaque peut se dérouler. Plusieurs possibilités s'offrent alors au hacker. Il peut tenter de subtiliser des données dans le système qu'il attaque en utilisant les outils de *Session Highjacking* , voler des mots de passe et des logins du système nécessaires à l'accomplissement de l'étape suivante. Elle consiste à cacher sa présence au sein du système en utilisant un RootKit. Enfin, après avoir modifié les fichiers qui lui sont nécessaires, il ne lui reste plus qu'à configurer une brèche ouverte en permanence afin de revenir ultérieurement dans le système.

Data Category	Types of data to collect
Network performance	• total traffic load in and out over time (packet, byte, and connection counts) and by event (such as new product or service release) • traffic load (percentage of packets, bytes, connections) in and out over time sorted by protocol, source address, destination address, other packet header data • error counts on all network interfaces
Other network data	• service initiation requests • name of the user/host requesting the service • network traffic (packet headers) • successful connections and connection attempts (protocol, port, source, destination, time) • connection duration • connection flow (sequence of packets from initiation to termination) • states associated with network interfaces (up, down) • network sockets currently open • whether or not network interface card is in promiscuous mode • network probes and scans • results of administrator probes
System performance	• total resource use over time (CPU, memory [used, free], disk [used, free]) • status and errors reported by systems and hardware devices • changes in system status, including shutdowns and restarts • file system status (where mounted, free space by partition, open files, biggest file) over time and at specific times • file system warnings (low freespace, too many open files, file exceeding allocated size) • disk counters (input/output, queue lengths) over time and at specific times • hardware availability (modems, network interface cards, memory)
Other system data	• actions requiring special privileges • successful and failed logins • modem activities • presence of new services and devices • configuration of resources and devices
Process performance	• amount of resources used (CPU, memory, disk, time) by specific processes over time; top "x" resource-consuming processes • system and user processes and services executing at any given time
Other process data	• user executing the process • process start-up time, arguments, file names • process exit status, time, duration, resources consumed • the means by which each process is normally initiated (administrator, other users, other programs or processes), with what authorization and privileges • devices used by specific processes • files currently open by specific processes

Files and directories	• list of files, directories, attributes • cryptographic checksums for all files and directories • accesses (open, create, modify, execute, delete), time, date • changes to sizes, contents, protections, types, locations • changes to access control lists on system tools • additions and deletions of files and directories • results of virus scanners
Users	• login/logout information (location, time): successful attempts, failed attempts, attempted logins to privileged accounts • login/logout information on remote access servers that appears in modem logs • changes in user identity • changes in authentication status, such as enabling privileges • failed attempts to access restricted information (such as password files) • keystroke monitoring logs • violations of user quotas
Applications	• applications- and services-specific information such as network traffic (packet content), mail logs, FTP logs, web server logs, modem logs, firewall logs, SNMP logs, DNS logs, intrusion detection system logs, database management system logs. Services specific information could be • for FTP requests: files transferred and connection statistics • for web requests: pages accessed, credentials of the requestor, connection statistics, user requests over time, which pages are most requested, and who is requesting them • for mail requests: sender, receiver, size, and tracing information; for a mail server, number of messages over time, number of queued messages • for DNS requests: questions, answers, and zone transfers • for a file system server: file transfers over time • for a database server: transactions over time
Log files	• results of scanning, filtering, and reducing log file contents • checks for log file consistency (increasing file size over time, use of consecutive, increasing time stamps with no gaps)
Vulnerabilities	• results of vulnerability scanners (presence of known vulnerabilities) • vulnerability patch logging

D'après [CERT 00-a]

CHAPITRE II PRESENTATION D'AAFID (AUTONOMOUS AGENT FOR INTRUSION DETECTION)

2.1. Introduction : Présentation du projet AAFID

Après avoir dressé un bref aperçu des concepts essentiels de la sécurité informatique, abordons à présent les différents aspects de l'IDS AAFID en débutant par la genèse du projet que mène par une équipe d'universitaires américains. Nous examinerons dans un premier temps l'architecture d'AAFID et ses divers composants[19]. Dans la seconde partie nous verrons les choix fondamentaux de conception posés par les concepteurs d'AAFID pour examiner, dans la dernière partie, les différentes options prises pour l'implémentation.

AAFID, qui signifie Autonomous Agents For Intrusion Detection, est un IDS développé au CERIAS, laboratoire de recherche de l'université américaine de Purdue sous la direction du professeur Eugene H. Spafford et de son assistant Diego Zamboni.[20] Ces derniers ont développé, un système de détection des attaques dont nous allons présenter dans ce chapitre l'architecture et le mode de fonctionnement.

La première architecture d'AAFID est proposée en 1994 par Spafford et Crosbie qui avancent l'idée d'utiliser des agents autonomes évoluant seuls automatiquement en utilisant de la *genetic programming* pour détecter les attaques. [SPA 00 : 551] Cette idée n'est pas implémentée comme telle mais l'idée d'utiliser des agents est gardée. Entre 1995 et 1996 l'architecture d'AAFID est créée au COAST Laboratory[21]. Elle sera une nouvelle fois modifiée pour devenir l'architecture que nous connaissons et allons étudier.

AAFID tourne sous les systèmes UNIX comme Solaris et Linux. AAFID2, dernière version disponible, a été entièrement implémenté en Perl 5 orienté objet[22]. Les auteurs avancent trois raisons pour motiver ce choix : la facilité d'élaborer et tester des prototypes, la portabilité et la grande flexibilité.

---

[19] Pour présenter le système AAFID nous utilisons la présentation réalisée par les auteurs, Eugene SPAFFORD et Diego ZAMBONI dans plusieurs articles et sur le site web du projet. Consulter la bibliographie pour avoir les références détaillées ([SPA 00] [BAL 98] [SPA 98] [AAF 99]).

[20] CERIAS = Center for Education and Research in Information Assurance and Security. Centre de formation et de recherché attaché à l'université de Purdue. Le CERIAS s'occupe de groupes de recherches en sécurité informatique. Le CERIAS est un des centres de recherches les plus performants en sécurité informatique au sens large. http://www.cerias.purdue.edu

[21] COAST (Computer Operation Audit and Security Technology) Laboratoire de recherche en sécurité informatique créé au sein même du CERIAS en 1987 par Eugene Spafford. Le COAST est devenu une véritable organisation à part entière en 1992. L'objet de ce groupe de recherche est d'étudier et mettre au point de nouvelles techniques de protection en sécurité informatique.
http://www.cerias.purdue.edu/coast/intrusion-detection/

[22] Nous distinguerons plus loin les différents langages utilisés lors des implémentations successives d'AAFID.

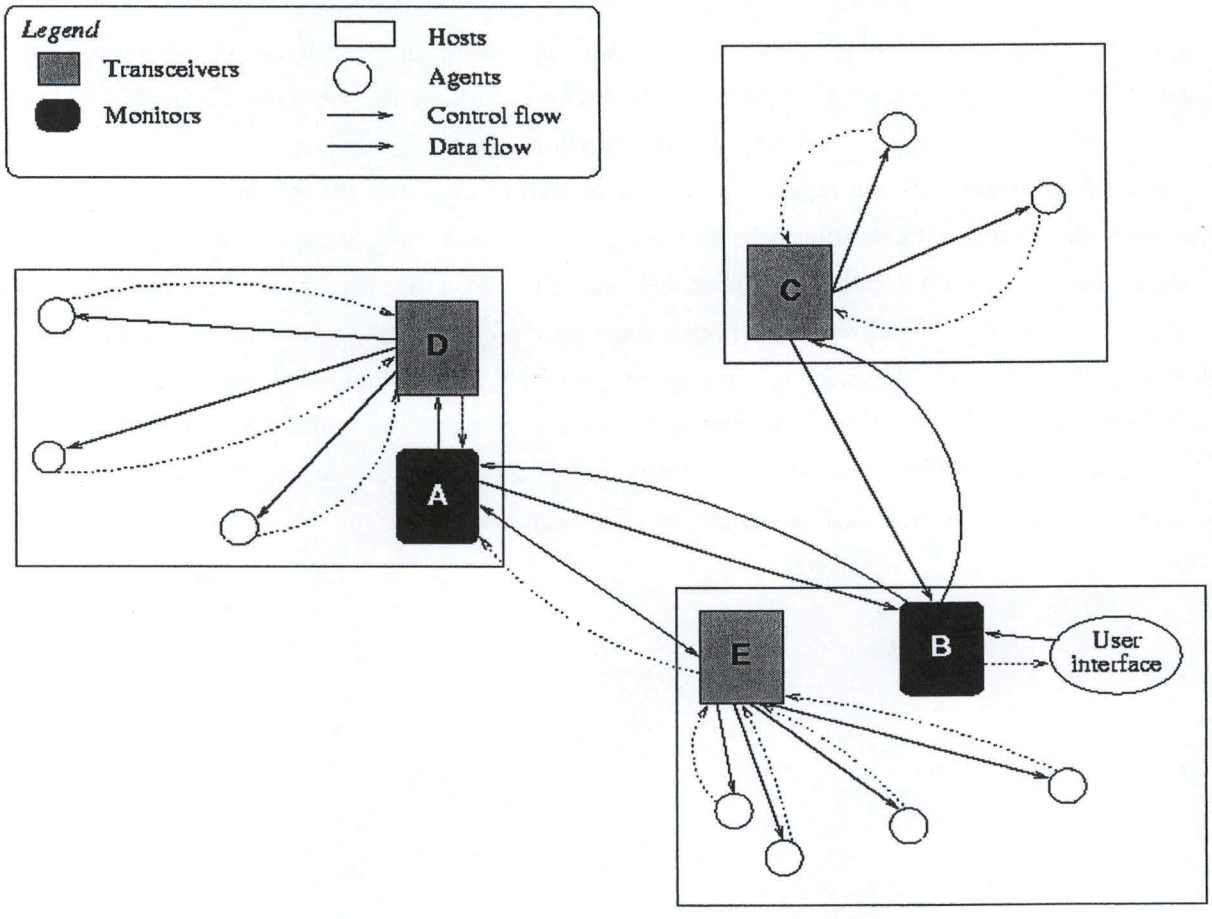
Avant d'aborder l'examen de l'architecture en tant que telle, voyons ce que les auteurs du projet entendent par Agents Autonomes acteurs principaux de l'IDS AAFID. Un Agent Autonome est un agent logiciel qui effectue une tâche précise de surveillance de la sécurité de son hôte. Ils sont qualifiés d'autonomes parce qu'ils peuvent fonctionner comme des entités à part entière indépendamment d'une superstructure. Ils peuvent ou non avoir besoin de données fournies par d'autres agents et acceptent des commandes envoyées par des entités de plus haut niveau.

Un agent, pour être efficace, doit présenter plusieurs caractéristiques. Il doit tout d'abord tourner continuellement pour ne pas manquer une tentative d'attaque. Sa résistance aux fautes et aux tentatives de subversions constitue également deux qualités importantes. Un agent doit utiliser un minimum de ressources du système pour ne pas détériorer les performances générales. Ensuite, il doit être configurable, adaptable et proportionné à l'échelle du système qu'il doit protéger.

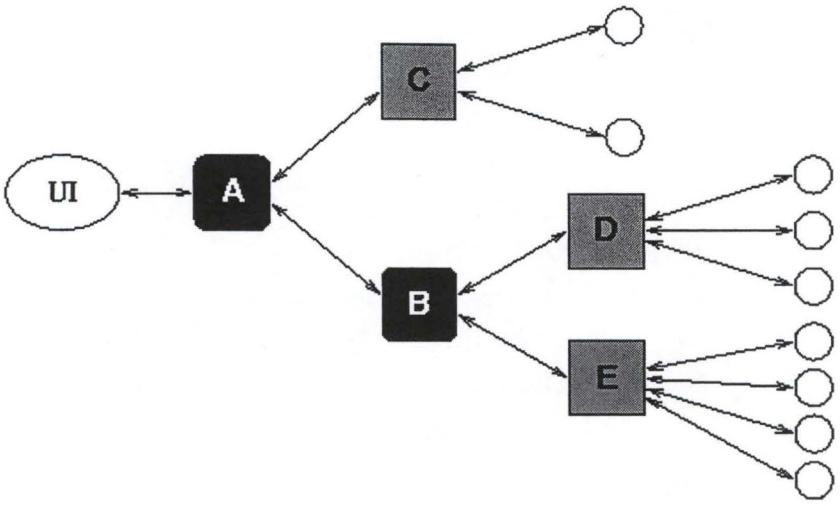
2.2. Architecture d'AAFID

2.2.1. Aperçu général de l'architecture

On distingue quatre composants dans l'architecture d'AFFID : les agents, les filtres, les moniteurs et les transmetteurs.[23] Les deux schémas ci-dessous présentent deux vues différentes de l'architecture. [BAL 98 : 5] La première (figure 1) montre la disposition physique des composants au sein du système AAFID ainsi que les flux de communication entre-eux. La seconde (figure 2) est une vision logique de l'architecture qui privilégie la hiérarchie de communication entre composantes.



[23] Nous utiliserons également le terme d'entité pour qualifier chacune de ces composantes.



Le système AAFID peut être installé sur plusieurs machines appartenant à un même réseau ou sur une seule machine. Chaque machine hôte peut contenir plusieurs ***agents*** qui surveillent chacun un type d'événements particuliers. Les agents peuvent utiliser des ***filtres*** afin d'obtenir des données nécessaires à leur fonctionnement. Tous les agents présents sur une machine transmettent leur rapport à un seul ***transmetteur***[24] qui est le seul sur cette machine à avoir une vision d'ensemble des divers agents et peut les commander ou les configurer différemment. Chaque transmetteur envoie son rapport à un ***moniteur*** qui gère à un niveau supérieur, plusieurs émetteurs/récepteurs placés sur plusieurs machines. Un moniteur a accès à toutes les données générées et est seuls capables d'effectuer des corrélations entre des événements localisés qui constituent une attaque. A noter que plusieurs moniteurs peuvent fonctionner en parallèle afin de limiter les risques d'erreurs ou de pannes. Enfin, le moniteur est le seul capable de fournir des informations à l'interface utilisateur.

[24] Un transmetteur ou contrôleur est une entité qui sert de relais communicationnel au sein d'AAFID se sont des émetteurs/récepteurs.

2.2.2. Composants de l'architecture d'AAFID

a) Agents

Les agents sont des entités qui tournent sur une machine et surveillent un aspect particulier de la sécurité du système. Un agent est programmé pour effectuer une tâche précise et générer, à période fixée, un rapport d'activité. Notons qu'un agent n'a pas la faculté de générer une alarme. En effet, seul un transmetteur ou, à fortiori, un moniteur pourront le faire. En outre, il n'y a aucune communication inter-agents au sein du système puisque toute l'information est centralisée par les émetteurs/récepteurs. Enfin, remarquons que l'architecture d'AAFID a été conçue pour que l'on puisse y ajouter très facilement de nouveaux agents. La seule condition réside dans le respect de la syntaxe fixée pour les commandes envoyées à l'agent et les messages générés par celui-ci.

b) Filtres

Les filtres sont, en fait, des fournisseurs de données pour les agents. Dans la première mouture de l'architecture d'AAFID, chaque agent se procurait seul des données. Au sein d'un système, plusieurs agents peuvent avoir besoin d'accéder aux mêmes données, par exemple un fichier de logs. Donc plutôt que d'implémenter dans chaque agent un analyseur de fichier, les concepteurs d'AAFID ont choisi de déléguer cette fonction aux filtres qui le font une fois pour tous les agents. Au sein d'AAFID, il n'existe qu'un seul filtre par type de données auprès duquel les agents se fournissent.

c) Transmetteurs ou contrôleurs

Ils constituent les seuls relais de communication disponibles entre les diverses machines et assurent deux grands rôles : le contrôle et le traitement des données. Afin de traiter les données, les émetteurs/récepteurs recueillent les divers rapports en provenance des agents tournant sur la même machine. Ils assurent le traitement de ces informations et transmettent ensuite ces données au(-x) moniteur(-s). Ils peuvent également renvoyer des commandes aux agents. En outre, ils contrôlent les agents placés sous leur responsabilité en les lançant ou les arrêtant et en assurant le relais entre le moniteur et les agents.

d) Moniteurs

Dans l'architecture d'AAFID, les moniteurs sont les entités du plus haut niveau. D'une part ils jouent le même rôle que les émetteurs/récepteurs au point de vue contrôle et traitement des données. Mais d'autre part ils se distinguent de ces derniers par la vue qu'ils ont de la totalité de l'IDS. Ils assurent donc le traitement des données en provenance des émetteurs/récepteurs et sont capables d'effectuer des corrélations entre les données qu'ils reçoivent et de détecter des problèmes qui impliquent plusieurs machines du réseau. De plus, étant donné que l'interface utilisateur d'AAFID est directement connectée à un moniteur, celui-ci devient le point d'entrée dans l'ensemble de l'IDS. Ainsi un moniteur pourra transmettre des ordres à d'autres moniteurs ainsi qu'à des émetteurs/récepteurs disséminés sur le réseau.

e) Interface homme/machine

L'interface homme-machine d'AAFID constitue le point de contrôle de l'ensemble du système. Celle-ci interagit directement avec un moniteur afin de surveiller l'état du système et de transmettre des commandes. Deux interfaçages sont dès lors possibles : graphique ou mode texte.

2.3. Choix de conception d'AAFID

2.3.1. De la première version d'AAFID à AAFID2

La version d'AAFID sur laquelle nous avons travaillé est la dernière des versions d'AAFID disponible. En effet, entre 1995 et 1996 la première version d'AAFId est implémentée sur base de la première version de l'architecture. Cette version est programmée en utilisant du C, Tcl/Tk et Perl. Le but de cette version consiste à tester simplement l'architecture élaborée. Cette version n'a jamais été rendue publique.

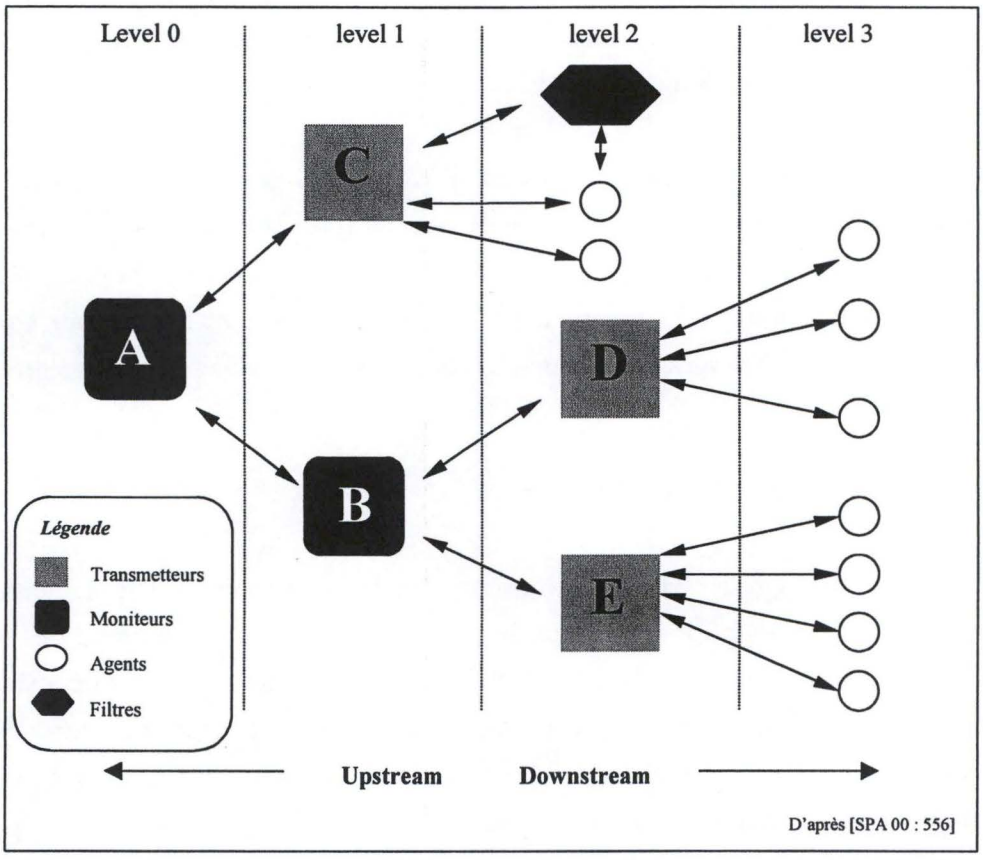
En 1997 débute la seconde implémentation d'AAFID qui devient donc AAFID2. Cette fois elle est entièrement réalisée en Perl afin de rendre l'application portable. Elle est terminée un an plus tard, en septembre 1998. Cette version ne comprend que quelques agents et est seulement testée sous Solaris. Ce prototype est complété puis testé sous Solaris et Linux pour être rendu public en septembre 1999. C'est sur cette dernière version que notre travail s'est concentré.

2.3.2. Les sept objectifs d'AAFID2

Le terme objectif signifie, dans le cas présent, choix ou directions d'implémentation. Il s'agit de distinguer les critères prioritaires et de second plan du projet. Le premier des objectifs poursuivis était simplement de *tester l'architecture* d'AAFID pertinente sur le papier mais jamais testée. Il fallait donc fournir un système que des organisations puissent utiliser et tester dans leur configuration de réseau afin de connaître les forces et les faiblesses. En second lieu, AAFID2 devait être *facile à utiliser.* Pour ce faire, les différents modules sont relativement indépendants permettant ainsi une utilisation plus aisée. Ensuite, l'accent a été mis sur la *facilité de configuration* et sur les possibilités d'*extension* du programme. Chaque module peut donc être très facilement configuré grâce à une standardisation des commandes. La *facilité d'installation* est le quatrième des objectifs. Etant donné la possibilité donnée de modifier en cours de route les composants d'AAFID, une installation facile s'imposait. Le cinquième objectif, constitue plutôt un choix d'implémentation. Il s'agit de ne pas accorder trop d'importance aux *performances* sans toutefois tomber dans l'excès inverse et mettre au point une application lente et ralentissant le système. Dans la même optique que le cinquième objectif, les concepteurs d'AAFID n'ont pas accordé la priorité à la *sécurité des agents* au sein de l'IDS. Enfin, dernier objectif, très important à leurs yeux, assurer une *infrastructure de développement* d'AAFID avec l'application. Ainsi ils ont fourni dans cette dernière version un module facilitant la création de nouveaux agents.

2.3.3. Hiérarchisation de la communication dans AAFID.

Les flux de communication entre les entités constituent un des aspects fondamentaux de l'application Pour les expliciter, nous allons reprendre le schéma proposé par E. Spafford (figure 3) [SPA 00 : 556]. Ce schéma reprend les flux de communication dans AAFID. Les flèches représentent les échanges de données et les commandes de contrôle.



A partir de ce modèle d'AAFID, Eugene Spafford et Diego Zamboni ont dérivé cinq définitions terminologiques.

- Définition 1 : (*Upstream, Downstream, above, below*) Dans le modèle ci-dessus, on peut identifier deux types de niveaux. Ceux situés près de la racine de la hiérarchie, sont dits en « amont » (*Upstream*) tandis que les niveaux près des feuilles de l'arbre seront en « aval » (*Downstream*). En outre, on dira qu'une entité se situe « au-dessus » (*above*) d'une autre entité si elle peut être atteinte par un chemin allant vers l'amont. Elle sera donc « en-dessous » (*below*) si elle peut être jointe en suivant un chemin vers l'aval. Par exemple A est au-dessus de B, C, D et E tandis que D et C sont en-dessous de A et B mais pas de C.

- Définition 2 : (Super-entités et sous-entités) Un sous-entité sera une entité qui se situe au-dessous d'une autre entité et vice-versa.

- Définition 3 : (Entités parents et enfants) Pour une entité donnée, une super-entité, située au niveau immédiatement supérieur sera considérée comme entité parent. L'entité située au niveau au-dessous sera par conséquent une entité enfant.

- Définition 4 : (*Up* et *down*) Cette terminologie concerne ici le sens dans lequel transitent les messages. Si une entité envoie un message à une entité du niveau supérieur on parlera de *Up*. Dans le cas contraire, on emploiera le terme *down*.

- Définition 5 : (Entité de contrôle) C'est une entité qui exerce un contrôle sur un certain nombre d'autres entités.

2.3.4. Fonctionnalités des diverses entités

a. Pour tous les types d'entités

Au sein d'AAFID2, on peut remarquer plusieurs caractéristiques communes à toutes les entités quel que soient leur type. La première caractéristique commune se situe au niveau de la très grande autonomie de ses entités. En effet, ses concepteurs ont voulu que chaque entité puisse être un programme pouvant être lancé et fonctionner indépendamment des autres entités. Il est donc possible avec AAFID2 de lancer une entité et dialoguer avec par commandes interposées. Bien entendu n'importe quelle entité doit également pouvoir être lancée et dirigée par l'entité supérieure qui la contrôle.

Ensuite, toutes les entités possèdent un identifiant unique ainsi qu'une brève description de leur rôle et leurs fonctionnalités. Elles doivent pareillement réagir à un ensemble fixé de messages qui induisent des comportements pré-programmés tels que s'arrêter, effectuer une recherche d'informations. Sur ce point, notons que l'on peut, pour chaque entité, redéfinir de nouvelles commandes afin de satisfaire à de nouvelles fonctionnalités.

Enfin, chaque entité doit pouvoir échanger des messages avec l'entité qui la contrôle et répondre adéquatement aux messages qui lui sont adressés. Cette caractéristique est valable pour une entité séparée, sur le plan physique, de l'entité qui la contrôle. AAFID2 a été conçu afin que l'on ait pas à se soucier de la communication entre entités situées sur des machines différentes.

b. Agents

Un agent est une entité qui récolte des informations bien précises sur le système qui l'héberge et analyse celle-ci afin déceler un éventuel problème de sécurité. Les agents ont un comportement global clairement établi. Voici l'algorithme proposé par les initiateurs d'AAFID [SPA 00 :558]. Les lignes en italique sont à définir par le concepteur de l'agent.

{Instantiation of the agent}
{*Instantiation-time initialization*}
Set event handlers
{Execution of the agent}
Run-time initialisation
Set more event handlers
Contact necessary filters
Enter Event loop.

Cet algorithme vaut pour tous les agents, excepté les lignes en italique qui sont spécifiques à un agent particulier. Toutes les autres fonctionnalités sont fournies par l'infrastructure afin de permettre la création de nouveaux agents assez rapidement.

c. Les filtres

Les filtres sont des entités qui servent à fournir des données aux autres entités par qui les sollicitent. Ils doivent pouvoir accéder facilement aux fichiers et autres sources de données du système qui les hébergent. Ce sont les seules entités autorisées à communiquer avec d'autres entités de même niveau (voir figure 3). Ils sont lancés par les transmetteurs mais reçoivent les commandes des agents à qui ils fournissent des données. Un filtre doit en outre être capable de fournir des informations à plus d'un agent.

d. Les transmetteurs

Le rôle principal d'un transmetteur réside dans le contrôle de tous les agents d'un hôte particulier. Pour ce faire ils doivent pouvoir communiquer avec le plus grand nombre d'entités qui les entourent. La majorité des échanges se fait avec les agents qui envoient leur message d'état ou des requêtes particulières. En outre, les transmetteurs doivent pouvoir répondre aux commandes d'un moniteur. Enfin les transmetteurs sont implémentés afin de faire suivre un message qu'ils ne parviennent pas à interpréter à une entité parent capable de la traiter.

e. Les moniteurs

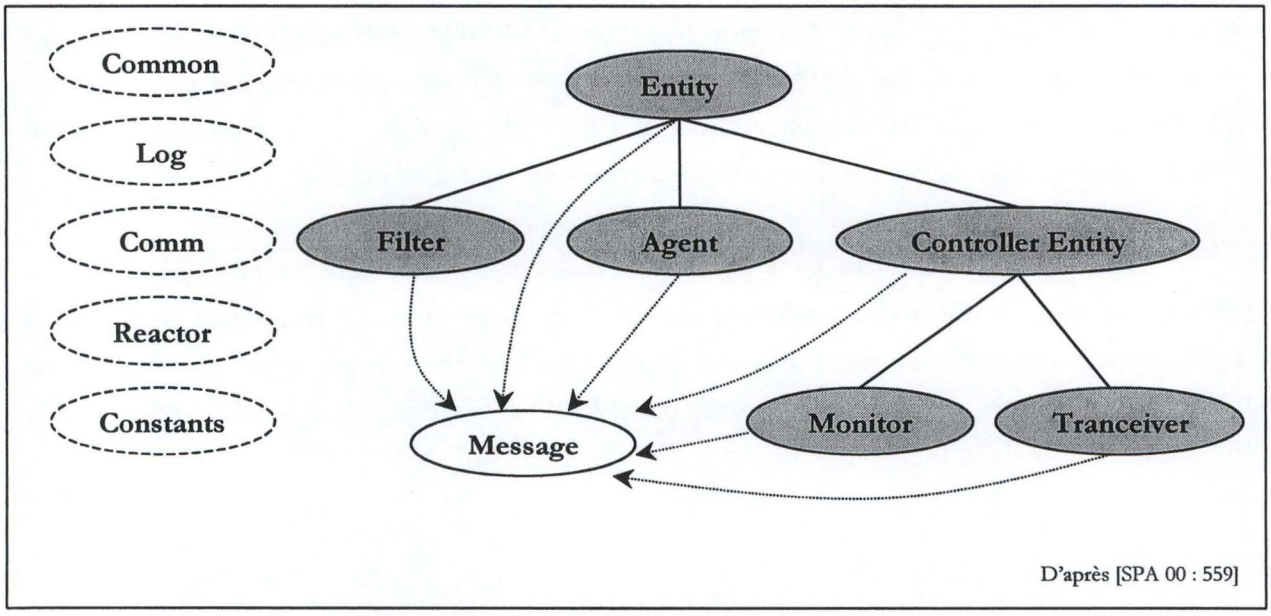
Dans l'implémentation d'AAFID2, un moniteur a les mêmes fonctionnalités qu'un transmetteur, c'est-à-dire le contrôle d'entités, à la différence près qu'il doit pouvoir démarrer et arrêter à distance d'autres entités situées sur d'autres machines. Ces entités peuvent être d'autres transmetteurs ou moniteurs avec lesquels il communiquera. Au point de vue des communications, un moniteur entre en relation directe uniquement avec des transmetteurs et d'autres moniteurs. Il ne communique pas avec les agents, souvent nombreux, afin de réduire le nombre d'entités qu'un moniteur dirige.

Par ailleurs, un moniteur doit surveiller l'établissement de nouvelles connexions établies par des entités distantes. Cette exigence est due au fait qu'un transmetteur ou un autre moniteur peuvent être lancés et doivent donc être pris en charge par un moniteur existant. Cette nouvelle entité doit contacter le moniteur et s'enregistrer auprès de celui-ci en lui envoyant un message de connexion.

Enfin, la fonction majeure des moniteurs réside dans la gestion globale de l'information que leurs sous-entités peuvent vouloir obtenir. Par information, il faut entendre ici des parties de codes qu'un transmetteur ne trouve pas sur la machine qui l'héberge.

2.3.5. La représentation d'AAFID2 en Orienté Objet

Nous allons à présent examiner la manière dont AAFID2 a été structuré dans son implémentation en Perl Orienté Objet. Les auteurs d'AAFID2 ont pris comme règle de toujours mettre comme préfixe, au nom d'une classe, le terme AAFID. Par exemple la classe agent sera identifiée par AAFID ::Agent. Toutes les sous-classes héritent des fonctionnalités de leur parent et peuvent les étendre. Le schéma suivant (figure 4) représente les diverses classes et les relations qui existent entre elles. Lorsque deux classes sont reliées par une ligne continue, il s'agit d'une relation d'héritage. Les lignes en pointillés marquent la notion d'utilisation d'une classe par une autre selon le sens de la flèche. Les classes qui appartiennent à la hiérarchie principale sont grisées, excepté la classe *Message* qui est à part. Les autres, dont les contours sont en pointillés, forment les classes auxiliaires qui contiennent des méthodes que les autres classes pourront utiliser.



D'après [SPA 00 : 559]

2.3.6. Les différents statuts d'une entité

Une entité d'AAFID peut adopter plusieurs statuts qui ont préalablement été définis. Les statuts dépendent de la gravité du problème éventuellement détecté par l'entité. Il faut distinguer ici la différence entre le statut global de l'IDS qui dépend de l'ensemble des statuts de chaque entité particulière. Le statut d'une entité est annoncé par le paramètre *status* accompagné d'un *message,* sorte de mise en texte du statut. Le statut possible s'échelonne de 0 pour « tout est normal » à 10 « extrêmement alarmant ».

2.3.7. Messages et commandes

Les messages sont échangés entre les diverses entités ainsi que le montre la figure 3. La structure d'un message se compose des éléments suivants : un type de message, un sous-type de message, un identifiant de la source, un identifiant de la destination, l'heure, les données. Cette structure est utilisée par toutes les entités au sein d'AAFID2 afin de représenter toutes les informations échangées. Le champ contenant le type de message détermine en réalité les informations que l'entité destination va trouver dans le champ données. Cette dernière pourra, après analyse du type, traiter correctement les données qui lui sont fournies.

Quant aux informations concernant l'émetteur et le destinataire du message, elles sont obligatoires et ne concernent exclusivement que ces deux entités particulières. L'heure est représentée en secondes comptabilisée depuis le 1 janvier 1970 afin de respecter les standards UNIX. Les messages et commandes utilisés par AAFID sont définis dans les deux tableaux à la fin de ce chapitre [SPA 00 :561-562].

2.4. Implémentation d'AAFID2

Les concepteurs d'AAFID2 ont posé plusieurs choix importants d'implémentation qui intéressent particulièrement la suite de notre travail. Outre le langage utilisé, Perl 5, nous allons examiner les points principaux de l'implémentation d'AAFID, c'est à dire les paramètres utilisés par les entités, le format des messages et la réaction à ces messages, les techniques de communication ainsi que les types d'événements et leur gestion.

2.4.1. Les entités et leurs paramètres

En Perl, les objets sont représentés par des références qui pointent vers des variables de type scalaire, ou un tableau, une chaînes de caractères, ou un tableau associatif (*hash)* [TIL 00 : 668]. En Perl Orienté Objet on représente les objets par une référence à un tableau associatif qui est une sorte de tableau dont les indices peuvent être n'importe quelle valeur. Les indices d'un tableau associatif peuvent donc être des chaînes de caractères. Un tableau associatif n'est donc pas assigné à une variable mais on y accède à travers une référence stockée dans une variable. Dans le cas d'AAFID2, la référence sera contenue dans la représentation de l'objet même.

Dans l'application qui nous occupe, chaque entité est un objet représenté par un tableau associatif qui contient les paramètres pour chaque entité spécifique. Les intitulés des paramètres forment les indices et le contenu de ces paramètres, les valeurs dont le type peut varier sans poser de problème. Etant donné que chaque instance de classe a son propre tableau associatif, chaque entité peut être dans un état propre totalement différent des autres entités. La classe *Entity* définit un certain nombre de méthodes qui sont utilisées afin de passer ou lire des valeurs aux paramètres.

2.4.2. Les messages

Les messages d'AAFID2 sont, en fait, des objets de la classe *Message* dont le rôle principal est de stocker les champs tels que nous les avons expliqués plus haut (type, sous-type,...) . A cela s'ajoute la capacité de convertir ces messages en un format propre à mettre sur le réseau. Une entité va garder les messages en tant qu'objets mais avant de les envoyer à une autre entité, elle va les transformer en une simple ligne de texte dont le format est le suivant : Type / sous-type / source/ destination / temps / données.

Lorsqu'une entité reçoit un message, donc une ligne, elle le parse et range les divers éléments du message dans un objet *Message* créé à cet effet. Tous ces champs sont des chaînes de caractères. La raison à l'origine de ce choix de type variable est motivée par la nécessité de communiquer facilement avec une entité via les canaux d'entrée/sortie traditionnels c'est à dire le clavier et l'écran.

De plus, à chaque type de message ou de commande correspond une action bien particulière. On peut ajouter à l'application de nouveaux types de messages. Si un nouveau message est reçu, une routine sera directement appelée comme méthode de l'entité ce qui implique qu'une référence à l'entité sera placé comme premier argument du message suivi par l'objet de ce message. Cette routine peut ou non retourner quelques chose. Dans l'affirmative ce sera un nouvel objet de type *Message*, et dans la négative, la valeur *undef*.

2.4.3. Communication inter-entités

Avant d'examiner les techniques choisies dans AAFID2, voyons les deux principales raisons qui ont inspiré ces choix : la transparence et l'indépendance vis-à-vis du système. En fait les entités ne doivent pas savoir comment elles communiquent entre elles, si une autre entité est sur la même machine ou non. Deux techniques de communication sont utilisées selon que la destination est sur la même machine ou sur une machine du réseau.

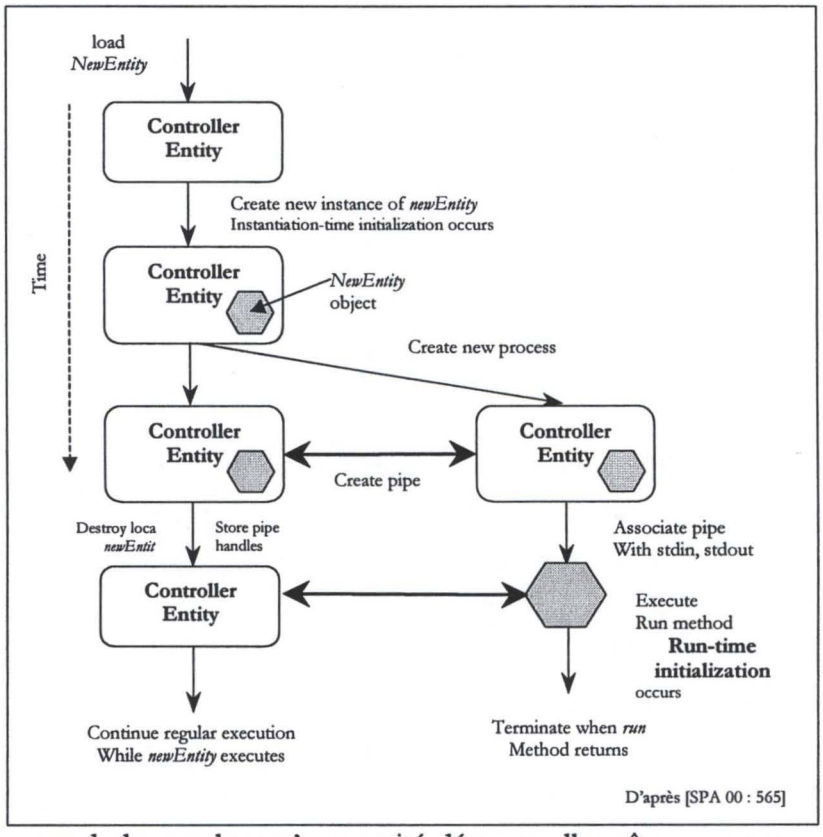
Pour communiquer avec une entité se trouvant sur le même hôte, la technique des tubes (*pipes*) est utilisée. Ce mécanisme de communication unidirectionnel suffit et présente l'avantage d'être utilisable sur n'importe quel système UNIX. Les communications entre entités distantes s'effectuent grâce au protocole TCP/IP. L'inconvénient de ces deux techniques réside dans l'absence de protection des données qu'elles véhiculent.

2.4.4. Les types d'événements et leur gestion.

Toutes les entités fonctionnent de la même manière en effectuant une boucle qui teste un certain nombre d'événements. Chaque entité connaît un ensemble prédéfini d'événements et est apte à les détecter et réagir de façon appropriée. Cette gestion des divers événements est gérée par la classe *Reactor*. AAFID2 distingue quatre types d'événements. Les *files handles* d'UNIX qui permettent d'accéder à un certain nombre d'objets du système tels que les tubes et les sockets. Une boucle aura pour mission de vérifier ces *files handles* afin de détecter l'écriture de nouvelles données dans le tube ou sur le socket. Le second type concerne les fichiers qui seront testés régulièrement afin de lire les informations qui ont pu y être écrite. Le troisième type est relatif à la notion de temps. Une entité peut être programmée pour réagir à un moment ou à intervalle donné. Enfin, les signaux générés par le système d'exploitation peuvent être considérés comme événements déclencheurs.

2.4.5. Chargement et exécution d'une entité.

Pour pouvoir être lancée, chaque entité doit disposer d'une méthode *run* qui sera le point de départ de son exécution. La fin de cette méthode *run* signifiera la terminaison de l'activité de l'entité. Pour gérer le moment où a lieu l'instanciation, la méthode *Init* est appelée au moment où l'objet est créé. Tandis que le moment du lancement est pris en charge par la méthode *RuntimeInit.* La première méthode sera appelée lorsque la nouvelle entité sera créée au sein du transmetteur



tandis que la seconde le sera lorsqu'une entité démarre elle-même son processus. Le schéma ci-dessus montre les deux types d'appel [SPA 00 : 565].

C'est la classe *Entity* qui fourni tout ce dont une entité a besoin pour fonctionner, soit les méthodes *run, Init, runtimeInit* et *Cleanup*. Cette dernière sert à effacer toutes les traces d'une exécution lorsque celle-ci est terminée. Toutes ces activités ont lieu dans le constructeur *new* défini dans la classe *Entity* et qui est hérité par les autres classes qui ne le redéfinissent pas.

Exécution en local

Il est également possible dans AAFID2 de procéder directement au lancement d'une entité seule. Cela est rendu possible grâce à un mécanisme de création automatique d'instance de la classe qui appelle la méthode run au moment de l'exécution de l'entité. Lorsqu'une entité est lancée seule, les messages qu'elle reçoit ou produit sont directement renvoyés au terminal de l'utilisateur qui l'emploie.

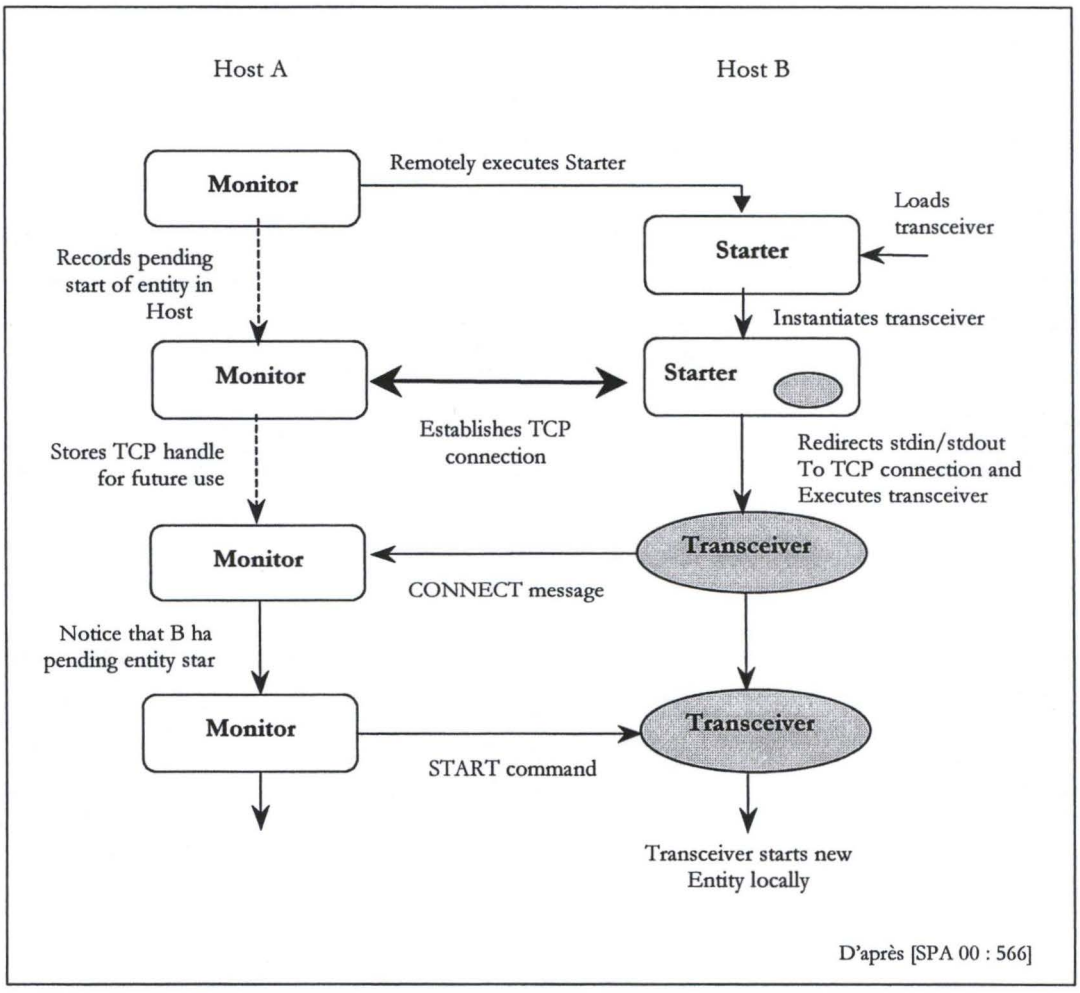
Cependant, lors d'une utilisation plus complète d'AAFID2, le mécanisme de lancement d'une entité par un autre programme, par exemple un transmetteur qui lance un agent, sera le suivant. Tout d'abord, la classe de l'entité est chargée grâce à la commande *use* de Perl. Ensuite, la méthode *new* crée une nouvelle instance de la classe. Un nouveau processus est créé afin de permettre l'exécution de la nouvelle entité. Pour pouvoir communiquer avec l'entité parent, deux tubes sont créés. Enfin, le processus fils exécute la nouvelle entité en invoquant la méthode *run*.

Exécution distante

L'exécution distante d'une entité est la dernière manière de procéder dans AAFID2. La classe *Monitor* fournit les différents éléments nécessaires à ce type de lancement. Plusieurs étapes successives sont accomplies pour ce faire. Mais avant de les citer, notons que toutes les entités sont lancées localement. Le transmetteur est lancé par le *Starter* et ensuite ce transmetteur crée une nouvelle entité. *Starter* est un programme qui établit la connexion réseau et est en relation directe avec le moniteur.

Le schéma suivant illustre les différentes étapes de l'exécution distante d'une entité. On distingue six étapes lors de ce type de lancement. Premièrement, le moniteur vérifie l'existence d'un canal de communication avec l'entité distante, moniteur ou transmetteur. Dans l'affirmative, il envoie un message pour demander le chargement de l'entité souhaitée.

Deuxièmement, il exécute le programme *Starter* sur l'hôte distant via ssh (Secure Shell). Ensuite, le moniteur signale qu'il attend une connexion de la part de l'hôte sur lequel il a exécuté le *Starter*.



Durant la quatrième étape, le *Starter* instancie un transmetteur sur l'hôte distant où il se trouve. Cela fait, le *Starter* contacte le moniteur sur un port défini afin d'établir une connexion TCP standard qui permettra d'échanger tous les messages. Ensuite le transmetteur peut démarrer et communiquer avec le moniteur grâce au canal de communication préalablement établi.

Sixième et dernière étape, lorsque le moniteur reçoit le message CONNECT du nouveau transmetteur, il peut lui envoyer à son tour une commande.

Messages utilisés dans AAFID2 d'après [SPA 00 :561]

Table 2
Standard message type defined in AAFID2

Subtypes	N/A
Description	Indicates that the message has not been initialised. This value can also be used in the message subtype field if that field has no specific value, or its value is irrelevant
Data field	N/A
Subtypes	CHILD, PARENT, FILTER, GUI
Description	Sent by an entity when it starts. It can be sent upstream to a parent entity, downstream to children, entities, and by filters and user interfaces. The subtype field indicates who is sending the message
Data field	Information about the entity. In AAFID2 it contains the entity's identifier and description.
Subtypes	CHILD, PARENT, FILTER, GUI
Description	Signals termination of the entity that sends the message. The purpose of this message is to allow the receiving entity to perform actions to maintain consistent internal information
Data field	Same as in CONNECT
Subtypes	Irrelevant
Description	The message contains a status update that the sender wants the recipient to have
Data field	Current status and descriptive message in the form of two subfields called `Status` and `Message` that contain the current numeric status and a descriptive message of the situation. The format of these fields is the Perl syntax for hash specification. For example `Status=>0, Message=>"No problems"`
Subtypes	Command name or RESULT
Description	Specifies a command that the receiving entities must execute. The subtype field contains the name of the command to execute. If a command produces a result, it will be sent back to the entity that requested the command in a message of type COMMAND and subtype RESULT. Each entity can define its own commands in addition to the standard set. See Table 3 for the list of standard commands defined in AAFID2
Data field	Named parameters to the command, in Perl hash-specification format (`Key=> Value`, separated by commas). In a RESULT message, it contains the result produced by the command (which should also be in the form of named parameters) plus a special parameter called `Command` that contains the name of the command that produced the result
Subtypes	Irrelevant
Description	Stops the execution of the entity, performing any necessary cleanup actions. This should usually involve at least sending a DISCONNECT message to any parent and children entities with the entity has communication
Data field	Irrelevant

Commandes utilisées dans AAFID2 d'après [SPA 00 :562]

Table 3
Standard commands defined in AAFID2

Parameters	None
Description	Has the same effect as a STOP message
Returns	N/A
Parameters	Code=>"*Perl code to execute*"
Description	Allows the execution of arbitrary code in the context of the entity that receives the message. This command is mostly for experimental purposes, for two reasons: it is easy to implement in Perl, but may be impossible to achieve in other languages; and it opens the possibility for security problems. The Code parameter contains the code to execute.
Returns	If the code executes without problems, does not produce a return value. If an error occurs, the result contains an ErrorMessage parameter that contains the description of the error.
Parameters	Parameter=>*Value* pairs, separated by commas
Description	Allows the specification of values for internal entity parameters, as described in Section 3.4.2
Returns	If the list of parameters and values contains an error, the command returns a description of the error. Otherwise it produces no return value.
Parameters	Params=>"*param1, param2,...*"
Description	Allows the retrieval of internal entity parameters. The Params parameters must be a string containing a comma-separated list of parameter names
Returns	A list of parameter name-value pairs containing the requested parameters and their value
Parameters	None
Description	Instructs the entity to produce an internal representation of its current state
Returns	A string representation of the entity (which is actually the current values of all its parameters), contained in the Me parameters. This representation can be used to examine the internal state of the entity, or possibly to initialise another entity to the same state

CHAPITRE III SPECIFICATIONS DES NOUVEAUX AGENTS

3.1. Introduction

Après avoir présenté l'architecture et le mode de fonctionnement d'AAFID2, nous allons à présent passer à la présentation et à la spécification des nouveaux agents que nous allons implémenter au sein d'AAFID2. Pour ce faire, nous allons dans un premier temps présenter sommairement les nouveaux agents et la raison de leur existence. A l'origine, on se trouve en présence d'un ensemble de tâches d'administration du système qu'il faut effectuer en utilisant soit des scripts ou des commandes introduites dans une console. Or ces tâches, pour la plupart, combinent les aspects administration du système et sécurité. Par conséquent, notre idée fut la suivante : pourquoi ne pas transformer ces tâches en nouveaux agents intégrés dans AAFID ?

Ces nouveaux agents sont au nombre de quatre et s'intitulent respectivement : SU, DiskSpaceCheck, FtpCheck et CoreDelete. Pour les spécifier, nous allons employer la méthode préconisée et utilisée par les auteurs d'AAFID. Cette dernière se décompose en quatre étapes successives. Il s'agit tout d'abord de détailler ce que l'agent va devoir détecter pour ensuite examiner la manière dont il va le signaler. Puis, lorsque les messages de sortie sont spécifiés, on examine la manière dont on pourra interagir avec l'agent, en d'autres termes, quelles sont les commandes qui seront nécessaires. Enfin, la dernière étape avant le codage consiste à réfléchir à la manière dont l'agent va être implémenté.

3.2. Les agents

3.2.1. Agent SU.

a. Que voulons nous détecter ?

L'agent SU, comme son nom l'indique, va s'occuper de surveiller l'utilisation la commande SU au sein du système. Cette commande lance un nouveau Shell et essaye, dans le même temps, de doter ce nouveau Shell d'un autre numéro d'utilisateur. Ce dernier est défini par le nom d'utilisateur qui apparaît comme paramètre. Par défaut, le nom d'utilisateur est celui de root. Si un utilisateur normal appelle cette commande sans paramètre, il essaye donc de se connecter au système en tant que root.

Cette commande constitue un point d'entrée éventuel dans le système pour une personne désireuse de s'accaparer les privilèges du root. Garfinkel et Spafford dans leur ouvrage consacré à la sécurité des systèmes UNIX soulignent l'importance que revêt la protection de cette commande [GAR 96 : 85]. Tout d'abord, le premier danger que nous avons évoqué est la tentative de s'accaparer les droits du root. Ensuite, ils insistent sur le mode de lancement de la commande SU. En effet, l'expérience montre que des pirates ont ajouté dans le chemin de recherche (search path) une pseudo commande SU qui est en réalité un cheval de Troie déguisé. De cette façon, un mot de passe peut ainsi être capturé ou un porte être ouverte dans le système. Par conséquent, la première mesure à prendre est de lancer la commande SU en tapant le chemin complet `/bin/su` de façon à invoquer la vraie commande[GAR 96 : 86].

Lorsque la commande SU est utilisée et peu importe son résultat, son exécution est inscrite dans le fichier *sulog*. Ce fichier se trouve dans le répertoire `/var/adm/sulog`. Ce fichier contient une suite de lignes dont le format est le suivant :

SU date heure résultat terminal utilisateur-nouvel utilisateur

La date est exprimée sous la forme du mois suivi du jour. L'heure est exprimée en heures et en minutes. Le résultat de l'exécution de la commande est représenté par le signe '+' si la commande a réussi, '-' si elle a échoué. Le paramètre terminal contient l'identifiant du terminal depuis lequel la commande a été lancée. Les deux derniers champs sont l'ID de l'utilisateur qui exécute la commande suivi par l'ID de l'utilisateur qu'il veut devenir.

L'agent que nous allons développer va donc devoir détecter les utilisations abusives de la commande SU. Toutes les tentatives ne seront pas traitées par cet agent. En effet, seules les tentatives de SU sans paramètres (donc root) sur le serveur seront traitées. C'est cet aspect qui nous a paru primordial au point de vue sécurité et performances du système.

Par conséquent, toutes les tentatives échouées de la commande SU seront traitées par l'agent.

b. Comment voulons-nous le signaler ?

L'agent que nous allons développer doit respecter les spécifications d'AAFID2 et par conséquent générer un type de sortie standard. Il s'agit d'un statut (*status*) et d'un message (*message*). Le statut va de 0 à 10 et croît proportionnellement à la gravité du problème que l'agent détecte.

L'agent va examiner le fichier sulog afin d'y lire les éventuelles tentatives échouées. Le statut va être incrémenté à chaque nouvelle ligne contenant une tentative échouée. Etant donné que l'agent peut être configuré pour ne vérifier une période de temps bien définie seules les tentatives réalisées dans un laps de temps proche incrémenteront le statut. Cela permet d'éviter une fausse alarme déclenchée si un utilisateur se trompe de terminal comme cela peut arriver.

Par conséquent, des tentatives répétées de SU vont générer un statut élevé signe d'un problème récurrent. L'agent va générer, lors de chaque contrôle, la sortie type : statut et message. En l'occurrence, le message contiendra le nombre de tentatives repérées et comptabilisées.

En outre, dès que le statut sera égal ou supérieur à 5, l'agent enverra un mail à l'administrateur pour lui signaler l'alerte. Ce mail contiendra un message reprenant toutes les informations sur la tentative échouée On pourra y trouver les informations contenues dans la ligne du fichier sulog posant problème afin d'identifier précisément l'origine de la tentative. Ce message se présentera sous la forme d'une chaîne de caractères comme l'exemple ci-dessous :

```
User=> guest9-root tried to become Root at date => 02/25 Hour=>
09 :31     From=> pts/5
```

Pour ce faire, l'installation d'un module Perl supplémentaire sera nécessaire. Il s'agit du module Mail ::Mailer disponible gratuitement sur Internet. Ce module permet d'intégrer dans les programmes Perl, les fonctionnalités de base de l'e-mail. Les éléments de base intéressants en l'occurrence sont les champs *From*, *To*, *Subject* et *Body*. Il nous suffisent dans le cadre des agents pour signaler une alerte avec quelques éléments de détails récoltés par l'agent et intégrés dans le corps du mail.

c. Comment allons-nous interagir avec l'agent ?

Mis à part les commandes de modification de paramètres déjà implémentée dans AAFID2, nous allons ajouter la commande SETFILE utilisée par plusieurs autres agents d'AAFID2. Bien que d'ordinaire le fichier à surveiller se situe dans /var/adm/, nous prévoyons une éventuelle différence de configuration. Par conséquent, grâce à cette commande, une modification de répertoire ne posera pas de problème puisqu'on pourra modifier le chemin du fichier grâce à cette commande.

La commande est la suivante :

```
SETFILE File => 'file'
```

d. Comment l'implémenter ?

La manière choisie pour l'implémentation de l'agent SU ressemble au niveau du principe à celle choisie par Diego Zamboni pour son agent LoginFailures. Cet agent fonctionne un peu sur le même principe que l'agent que nous allons créer. Il analyse un fichier de logs.

Le principe est le suivant. L'agent effectue un contrôle toutes les X secondes (*CheckPeriod*) et couvre des périodes de surveillance d'une durée Y (*SumPeriod*). Ainsi l'agent rapporte le nombre de tentatives de login infructueuse qui ont eut lieu durant la période Y. Nous allons procéder de la sorte pour l'agent SU en distinguant fréquence des contrôles et période contrôlée.

3.2.2. Agent DiskSpaceCheck

a. Que voulons nous détecter ?

L'agent DiskSpaceCheck comme son nom l'indique va s'occuper de surveiller l'espace disque occupé par les utilisateurs du système. En l'occurrence, l'aspect sécurité va de pair avec un aspect plus pratique d'administration du système. En effet, le volet sécurité est très présent puisque certaines attaques consistent à lancer des processus dont la taille ne fait que croître afin de saturer volontairement l'espace disque. D'un autre côté, la gestion de l'espace disque partagé entre les utilisateurs est un aspect important de l'administration d'un système.

Cet agent va être affecté très concrètement à la surveillance des disques de l'Institut d'informatique mais doit être souple afin de permettre la surveillance d'autres systèmes de fichiers différemment configurés. L'agent surveillera l'espace occupé sur les disques, signalera tout dépassement éventuel de l'espace autorisé.

b. Comment voulons-nous le signaler ?

Comme nous l'avons expliqué pour l'agent précédent nous respectons les spécifications d'AAFID2 en implémentant un agent qui génère des messages de sortie standards. A cet effet nous allons devoir distinguer plusieurs types de situations courantes dans l'utilisation de l'espace disque par plusieurs utilisateurs.

Dans la situation à laquelle nous sommes confrontés, il nous faut distinguer deux types d'utilisateurs : le personnel de l'institut (*staff*) et les étudiants (*students*). Cette distinction a lieu d'être puisque le personnel a droit à 300 Mo d'espace disque tandis que les étudiants sont limités à 100 Mo.

La solution la plus souple consiste à utiliser un agent par catégorie d'utilisateur à surveiller. La différence résidera au niveau du paramétrage de l'agent.

Nous avons opté pour un fonctionnement basé sur la notion de limite. En effet, une catégorie d'utilisateurs a droit à un espace bien défini ce qui va correspondre à la limite. Un utilisateur qui respecte cette limite est ignoré. Par contre l'agent va examiner tous les cas de dépassement selon leur importance. L'idée qui sous-tend le fonctionnement de l'agent se base sur l'idée qu'un dépassement léger est dû à une négligence sans préjudice donc il s'agit juste de la signaler. Par contre un dépassement important peut être causé volontairement afin de saturer le système. Donc ce type de dépassement sera pris en compte et signalé de manière plus importante. Ainsi nous avons distingué quatre cas de figure (excepté le cas de respect de la limite) :

1. Entre la limite et strictement inférieur à 2* la limite fixée
2. Entre 2* la limite et strictement inférieur à 4* la limite fixée
3. Entre 4* la limite et strictement inférieur à 10* la limite fixée
4. Plus de 10* la limite fixée

c. Comment allons-nous interagir avec l'agent ?

L'agent qui va vérifier l'espace disque devra recevoir un minimum de commandes de la part de l'utilisateur. Il faudra que l'utilisateur puisse indiquer à l'agent le répertoire à surveiller. L'introduction de cette commande rend possible l'utilisation de plusieurs agents du même type mais affectés à la surveillance de différents répertoires sensibles du système.

Par conséquent, nous aurons la commande suivante :

```
SETPARAMS me - - DirectoryTocheck =>'directory'
```

Cette commande nous permet d'ajouter un nouveau chemin à contrôler par l'agent. Lors de son appel et son exécution l'ancien chemin que l'agent surveillait est remplacé par celui passé en arguments. S'il n'existe pas l'agent retourne une erreur. Une commande pour effacer le chemin n'a pas raison d'être puisque cet effacement a lieu lorsqu'un nouveau chemin est spécifié.

d. Comment l'implémenter ?

Cet agent va devoir contrôler en détail un ensemble de répertoires et de sous-répertoires, ce qui prend généralement un temps assez important. Par conséquent, son fonctionnement va s'inspirer de l'agent précédent mais avec certaines nuances. L'agent aura un paramètre *CheckPeriod* que l'on pourra modifier. Le contrôle s'effectuera donc tous les X temps avec X = à *CheckPeriod.*

L'important ici est de trouver une fréquence de contrôle qui, d'une part, n'affecte pas les performances du système mais, d'autre part, garantit la sécurité du système. Ce point reste à déterminer et à justifier sur base de tests futurs.

3.2.3. Agent FtpCheck

a. Que voulons nous détecter ?

FTP qui signifie *File Transfer Protocol* permet aux utilisateurs de télécharger des fichiers entre deux machines. Ce système existe aussi bien sous les OS Microsoft Windows que sous UNIX, Solaris, Linux. Sous UNIX-Solaris, ce qui nous intéresse en l'occurrence, l'implémentation FTP distingue d'une part *ftp* qui est le client et */usr/sbin/in.ftpd* qui est le serveur. [GAR 96 : 487]. L'application utilise le port TCP n°21 pour échanger les commandes et elle assigne un port supérieur à 1024 pour l'échange de données.

Un serveur FTP peut être configuré afin d'accepter les connexions anonymes avec l'extérieur. Beaucoup d'institutions universitaires et privées configurent leur serveur FTP de la sorte afin de faciliter les échanges. Par conséquent, sur ces serveurs FTP, on peut venir télécharger des fichiers mais également en déposer.

C'est ce dernier aspect qui pose problème. En effet, depuis plusieurs années déjà les gestionnaires de ces serveurs ont remarqué que leurs serveurs devenaient des lieux d'échanges de *warez* qui sont en fait des programmes, jeux, fichiers musicaux et images piratés donc totalement illégaux [GAR 96 : 493].

Une des solutions trouver pour limiter les dégâts a été de limiter la possibilité de dépôt à un seul répertoire ouvert en écriture et donc plus facile à contrôler. C'était sans compter les astuces utilisées par la partie adverse, comme créer des répertoires invisibles dont le nom est composé d'espaces ou d'autres caractères astucieux.

Garfinkel et Spafford proposent une solution à ce problème [GAR 96 : 493]. Cette solution efficace limite toutefois la portée du serveur FTP en dépôt. Ils préconisent tout d'abord de créer un répertoire libre en écriture et pas en lecture appartenant au root et dont le mode est 1733. Ceci permet à n'importe qui de venir déposer mais rend impossible à un autre utilisateur de lister le contenu du répertoire. Ensuite, il faut limiter le dépôt par utilisateur à un certain nombre de mégabytes. Enfin, mettre au point un script qui, lancé périodiquement, va aller copier le contenu vers un autre répertoire accessible par FTP avec identification.

Une autre solution que nous allons utiliser dans le cadre de la mise au point de ce nouvel agent, consiste à travailler avec le fichier xferlog. Ce fichier contient en fait toutes les informations relatives aux connexions FTP enregistrées par le serveur FTP. Ce fichier se situe en général dans /var/adm/xferlog. Ce fichier de logs est constitué des éléments suivants [GAR 96 : 304]

Current-time; transfer-time; remote-host; file-size; filename; transfer-type; special-action-flag; direction; access-mode; username; service-name; authentication-method; authenticated-user-id.

Ces éléments représentent dans l'ordre, la date et l'heure du transfert, le nom de l'hôte distant à l'origine du transfert, la taille et le nom du fichier transféré. On trouve ensuite le mode de transfert (ASCII ou Binaire) puis un flag utilisé pour signaler si le fichier est compressé (C) ou non compressé (U) ou est une archive tar (T). La direction de transfert est ensuite spécifiée : o pour sortant (outgoing) et i pour entrant (incoming). Enfin, le dernier paramètre indique le type d'utilisateur connecté soit anonyme (a), invité (g) ou local (r).

Sur cette base nous allons devoir mettre au point un agent qui examine fréquemment ce répertoire et génère un message selon l'état du répertoire ouvert en écriture.

b. Comment voulons-nous le signaler ?

L'agent examinera régulièrement le fichier xferlog afin de détecter un éventuel dépôt de fichiers. Sur base de cette première constatation, il devra analyser plus en profondeur l'origine du dépôt et la taille des fichiers déposés. Il devra sur cette base générer son message de sortie (statut + message).

L'agent va donc examiner en cas de dépôt l'origine du dépositaire. A ce stade, deux cas de figure se présentent : soit le dépositaire provient des facultés (FUNDP) et donc pas de problème, soit il est extérieur à l'institution. C'est ce second cas de figure qui nous intéresse. Nous avons défini une taille limite de fichier s'élevant à 50 Mo. Si quelqu'un de l'extérieur vient déposer un fichier dont la taille est supérieure ou égale à cette limite, le statut de l'agent est incrémenté d'une unité afin de signaler cet état de fait. Si jamais l'agent atteint un statut supérieur ou égal à 5 un e-mail sera envoyé à l'administrateur système afin de lui signaler la situation. En outre l'agent fournira en sortie son statut et un message égal au statut.

c. Comment allons-nous interagir avec l'agent ?

L'agent recevra les commandes habituelles de configuration implémentées dans AAFID2. Nous ne voyons pas quelles autres commandes seraient nécessaires.

d. Comment l'implémenter ?

Etant donné que l'agent va analyser un fichier de logs, il peut fonctionner exactement sur la même base que l'agent SU. Il va donc vérifier le fichier tous les X temps (*CheckPeriod*) et cela pour une période de temps limitée (*SumPeriod*) cela afin d'éviter une redondance dans ses analyses et une perte de performances du système.

La fréquence de ces contrôles sera fixée par l'administrateur système.

3.2.4. Agent CoreDelete

a. Que voulons nous détecter ?

Ce dernier agent présente un aspect différent des autres agents car il ne comporte aucun aspect ayant directement trait à la sécurité du système. Cet agent effectue en réalité une tâche de maintenance du système. Son intégration au sein d'AAFID nous a paru intéressante au regard des nombreux avantages que présente l'utilisation de cet IDS.

La tâche de cet agent va donc différer quelque peu des diverses tâches que nous avons pu examiner jusqu'ici. L'agent CoreDelete va se charger de rechercher et d'effacer périodiquement les fichiers `core` des partitions des utilisateurs. Un fichier *core* est un fichier généré par le système d'exploitation lorsqu'un processus se termine suite à la réception d'un signal particulier. Le fichier core est alors créé dans le répertoire dans lequel le processus terminé tournait.. Ce fichier contient des informations sur l'état de la mémoire au moment où le processus tournait. L'inconvénient provient du nombre de fichiers *core* parfois générés et de leur taille variable. L'accumulation de ces fichiers contribue à réduire l'espace disque disponible. [GAR 96 : 872-873]

b. Comment voulons-nous le signaler ?

A la différence des autres agents, nous signalerons l'action de l'agent CoreDelete uniquement par l'envoi d'un mail à l'administrateur système lorsque la tâche sera terminée afin de l'avertir.

c. Comment allons-nous interagir avec l'agent ?

Nous ne prévoyons pas d'interaction particulière avec cet agent.

d. Comment l'implémenter ?

Nous allons l'implémenter de manière à ce que l'agent effectue périodiquement cette tâche de recherche et d'effacement. Pour ce faire nous maintenons le paramètre *CheckPeriod* afin de configurer à notre gré cette fréquence de travail.

CHAPITRE IV MODE D'UTILISATION ET TESTS DES NOUVEAUX AGENTS

4.1. Introduction

Une fois définis et implémentés (les codes et commentaires des agents suivent ce chapitre), il s'agit d'examiner la manière dont on peut utiliser chacun des agents grâce aux diverses commandes disponibles dans AAFID2. Cet aspect traité, nous examinerons les résultats des tests effectués afin de nous familiariser aux sorties générées par les quatre agents.

4.2. Utilisation des agents

Nous allons commencer cette partie en examinant la manière dont on utilise AAFID2 et ses agents d'une manière assez générale. AAFID2 a été conçu de manière à permettre trois modes d'exécution pour un agent. La première manière consiste à lancer un moniteur qui pourra lancer à son tour un agent ou un contrôleur. La seconde est de lancer l'agent par l'intermédiaire d'un contrôleur et la dernière méthode, que nous allons utiliser, lance un agent seul, sans aucune entité au-dessus de lui.

4.2.1. Le lancement

Pour lancer une entité d'AAFID (agent, moniteur ou contrôleur) la commande simple est la suivante :

```
Perl –w Entity.pm
```

Une fois la commande exécutée, elle va produire un message de retour dont voici la structure: **CONNECT CHILD <id> - <time> <class> <description>** L'élément <id> va représenter l'identifiant de l'entité formé du nom de la machine hôte, du nom de la classe de l'entité, du numéro de version et du numéro d'instance. L'élément <time> représente sous forme numérique l'heure du système tandis que <description> reprend la description interne de l'agent que l'on a défini dans le paramètre *Description* de ce dernier. Les messages qui vont suivre auront tous la structure que nous avons présentée dans le chapitre consacré à la présentation.

Le lancement d'un moniteur est donc très simple. En outre, on peut demander au moniteur ou au contrôleur, et ce, à tout moment, de renvoyer toutes les entités placées sous ses ordres grâce à la commande **command listentities** :

```
bash-2.03$ cd /usr/local/aafid2/classes/AAFID
bash-2.03$ perl -w Monitor.pm
CONNECT CHILD backus:Monitor:1.26:00 - 989047184 AAFID::Monitor Base Monitor Cla
ss
command listentities
COMMAND RESULT backus:Monitor:1.26:00 - 989048002 'NumEntities' => '0','Entities
' => '','Command' => 'LISTENTITIES'
stop
DISCONNECT CHILD backus:Monitor:1.26:00 - 989048009 STOP command or message
bash-2.03$
```

Le lancement d'un contrôleur s'effectue de la même manière :

```
bash-2.03$ cd /usr/local/aafid2/classes/AAFID
bash-2.03$ perl -w PlainTransceiver.pm
CONNECT CHILD backus:PlainTransceiver:1.09:00 - 989047255 AAFID::PlainTransceive
r Simple Transceiver
command listentities
COMMAND RESULT backus:PlainTransceiver:1.09:00 - 989048025 'NumEntities' => '0',
'Entities' => '','Command' => 'LISTENTITIES'

stop
DISCONNECT CHILD backus:PlainTransceiver:1.09:00 - 989047259 STOP command or
message
bash-2.03$
```

Enfin, le lancement d'un agent se différencie uniquement par la génération de son message de sortie :

```
bash-2.03$ cd /usr/local/aafid2/classes/Agents
bash-2.03$ perl -w agent3.pm
CONNECT CHILD backus:CoreDelete:1.02:00 - 989047328 CoreDelete Delete all Core f
iles in differents directories
STATUS_UPDATE NOTYPE backus:CoreDelete:1.02:00 - 989047330 Status => 0, Message
=> '0'
stop
bash-2.03$
```

Ainsi que l'illustrent ces trois exemples, n'importe quelle entité peut être arrêtée grâce à la commande **stop**.

4.2.2. Les commandes propres aux agents

Examinons à présent les commandes que nous allons pouvoir utiliser avec nos agents. Ces commandes sont en réalité de deux types. Tout d'abord, les commandes propres à tous les agents. Par exemple, la commande permettant de modifier un ou plusieurs paramètre(-s) de l'agent.

```
1) Command set_params me - - DirectoryToCheck => '/var/adm/sulog'
2) Command set_params me - - CheckPeriod => 10
3) Command set_params me - - DirectoryToCheck => '/var/adm/sulog', CheckPeriod
 => 10
```

La structure simple de cette commande permet de modifier à n'importe quel moment un des paramètres de l'agent à condition bien sûr qu'il existe dans l'agent. Comme le montre cet exemple, plusieurs modifications peuvent être effectuées en une seule commande.

Il existe, en outre, la commande inverse qui permet de demander à l'agent de renvoyer le contenu d'un de ses paramètres à condition bien sûr que le paramètre passé en argument à la commande existe. L'agent retourne le message **COMMAND RESULT** .

```
Command get_params me - - params => 'CheckPeriod'

COMMAND RESULT host:AgentName:1.05.0 me 885371551
command="GET_PARAMS", CheckPeriod=>10
```

Une autre commande, **dump_yourself**, est très intéressante parce qu'elle permet de connaître l'état de l'agent.

Command dump_yourself

Elle va renvoyer la représentation interne de l'agent, soit tous ses paramètres et variables au moment où la commande est appelée. Elle permet de mesurer l'évolution de l'agent lors de son exécution ou de détecter les bugs.

Le second type de commande regroupe toutes les commandes spécifiques aux divers agents. Celles-ci ont été implémentées au sein des agents pour satisfaire des exigences d'utilisations que les commandes de base ne permettaient pas. L'utilisation d'une commande définie dans un agent est très standardisée.

COMMAND *name* me - - Arg1 => Value1

COMMAND SETFILE File => '/var/adm/xferlog'

La première ligne illustre la syntaxe standard et la seconde montre l'utilisation de la commande SETFILE définie dans deux de nos agents.

4.3. Tests et résultats

Nous avons testé chacun de nos agent préalablement installés sur les serveurs de l'institut. Concrètement, les agents SU, DiskSpaceCheck et CoreDelete sont installés sur Bacchus tandis que l'agent FtpCheck va tourner sur Babbage qui abrite le serveur FTP de l'institut.

4.3.1. Agent SU

Afin de vérifier le bon fonctionnement de l'agent, nous avons procédé dans un premier temps à des tests sur un fichier *SULOG* « artificiel » c'est-à-dire présentant une structure identique au véritable fichier qui d'ordinaire est fermé, même en lecture, aux utilisateurs du système. A partir du moment où l'agent réagissait positivement lors de ces tests, nous sommes passés aux véritables tests sur le fichier *var/adm/sulog* dont les droits ont étés modifiés pour l'occasion.

```
bash-2.03$ perl -w SU.pm
CONNECT CHILD backus:SU:1.02:00 - 989327439 SU Check for repeated SU attempts
STATUS_UPDATE NOTYPE backus:SU:1.02:00 - 989327439 Status => 0, Message => '0'
STATUS_UPDATE NOTYPE backus:SU:1.02:00 - 989327740 Status => 2, Message => '2'
STATUS_UPDATE NOTYPE backus:SU:1.02:00 - 989328039 Status => 0, Message => '0'
STATUS_UPDATE NOTYPE backus:SU:1.02:00 - 989328339 Status => 0, Message => '0'
stop
DISCONNECT CHILD backus:SU:1.02:00 - 989328347 STOP command or message
bash-2.03$
```

Pour ces tests, nous avons configuré l'agent pour qu'il vérifie le fichier toutes les cinq minutes. Le résultat ci-dessus montre une alerte détectée lors de la seconde vérification. Le statut est alors incrémenté de 2 unités car deux tentatives ont été repérées. On retrouve dans le fichier *var/adm/sulog* les deux lignes à l'origine de l'alerte.

```
SU 05/08 15:11 - pts/39 salexand-root
SU 05/08 15:11 - pts/39 salexand-root
```

La difficulté majeure rencontrée lors du test a été le choix du paramètre *CheckPeriod* le plus approprié. Comme nous avons ajouté l'envoi d'un e-mail en cas d'alerte importante il ne faut pas que l'agent reste trop longtemps bloqué à un statut et envoie à chaque passage le même mail. Par conséquent, nous avons testé et fixé une fréquence 5 minutes entre les contrôles et une *SumPeriod* de 6 minutes.

4.3.2. Agent DiskSpaceCheck

Les premiers tests effectués avec cet agent se sont limités à une partition (*/home/rouge/students/salexand/students/*') dans laquelle nous avons créé plusieurs répertoires de tailles variables ayant des noms d'utilisateurs. Ensuite, nous avons configuré notre agent pour qu'il se limite à la vérification de ce répertoire en donnant à son paramètre *DirectoryToCheck* la valeur */home/rouge/students/salexand/students/*. De plus, pour ne pas occuper une place démesurée sur les disques de l'institut, nous avons configuré *LimitSize* à 2000 Kb. Ces tests se sont avérés concluants puisque l'agent détectait les utilisateurs en défaut et les ajoutait correctement à la liste envoyée par mail. Afin de vérifier si l'agent n'oubliait personne, nous avions préalablement, depuis une simple console, redirigé dans un fichier texte le résultat de la commande utilisée par l'agent.

Les tests suivants ont été réalisé sur */home/rouge/students/* avec des paramètres réalistes. La limite fixée était de 30Mo.

```
bash-2.03$ perl -w agent1.pm
CONNECT CHILD backus:DiskSpaceCheck:1.02:00 - 989563493 DiskSpaceCheck Check for
Disk Space
exceeding
STATUS_UPDATE NOTYPE backus:DiskSpaceCheck:1.02:00 - 989563519 Status => 7,
Message => 'USERS WITH AT LEAST 10*LIMIT OF DISK SPACE  '
stop
DISCONNECT CHILD backus:DiskSpaceCheck:1.02:00 - 989563522 STOP command or
message
bash-2.03$
```

4.3.3. Agent FtpCheck

Les tests de cet agent ont été réalisés en deux parties. Premièrement, de manière artificielle, en fournissant à l'agent une copie du véritable fichier *var/adm/xferlog* que nous avions copié sur une partition. Le but de ce test était de vérifier si l'agent lisait bien le contenu du fichier et détectait correctement les dépôts en discernant leur origine sur base du modèle (`/fundp|138.48/`) établi. Ces tests se sont révélés très concluants puisque tous les dépôts extérieurs étaient détectés par l'agent.

Dans un second temps, nous avons installé l'agent sur Babbage afin qu'il vérifie le véritable fichier *xferlog*. Le problème que nous allions rencontrer pour ces tests se situait au niveau de l'adresse du dépositaire. Logiquement, l'agent détecte les dépôts réalisés de l'extérieur des facultés mais pour des raisons de facilités de tests, nous ne pouvions pas à la fois lancer l'agent à l'institut et effectuer des dépôts depuis l'extérieur. Par conséquent, nous avons modifié le critère sélection pour que les dépôts provenant de l'intérieur de l'institution soient détectés. Cette modification opérée et le paramètre *CheckPeriod* fixé à 5 minutes, les essais ont pu commencer. Enfin, pour ne pas occuper inutilement de l'espace disque, nous avons configuré l'agent pour qu'il détecte les petits transferts (1Mo) et envoie un e-mail dès que son statut est supérieur ou égal à deux.

```
bash-2.03$ perl -w agent2.pm
CONNECT CHILD babbage:FtpCheck:1.02:00 - 989394704 FtpCheck Check FTP suspect
incoming
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989394704 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396054 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396204 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396354 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396504 Status => 1, Message
=> '1'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396654 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396804 Status => 2, Message
=> '2'
stop
DISCONNECT CHILD babbage:FtpCheck:1.02:00 - 989396885 STOP command or message
bash-2.03$
```

Lors de ce premier test, trois dépôts de fichiers ont été réalisés. Tous trois ont été détectés par notre agent qui, lors des deux derniers (statut = 2), a envoyé un mail pour signaler la situation. Le test suivant est presque identique hormis le fait que trois dépôts sont effectués à la suite.

```
bash-2.03$ perl -w agent2.pm
CONNECT CHILD babbage:FtpCheck:1.02:00 - 989396977 FtpCheck Check FTP suspect
incoming
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989396977 Status => 0, Message
=> '0'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989397127 Status => 3, Message
=> '3'
STATUS_UPDATE NOTYPE babbage:FtpCheck:1.02:00 - 989397278 Status => 0, Message
=> '0'
stop
DISCONNECT CHILD babbage:FtpCheck:1.02:00 - 989397394 STOP command or message
bash-2.03$
```

4.3.4. Agent CoreDelete

Pour commencer nous avons testé cet agent sur une partie limitée des répertoires de */home/rouge/students/* . Pour ce faire, nous avons copié six fichiers *core* dans divers sous-répertoires d'une partition à laquelle l'agent avait les droits d'accès. Nous avons ensuite exécuté l'agent avec comme le paramètre *DirectoryToClean* égal à *'/home/rouge/students/salexand/'*. L'agent a effectué sa tâche et a envoyé le mail de confirmation. Après vérification, tous les fichiers *core* avaient disparus des divers endroits où ils avaient été copiés.

```
bash-2.03$ perl -w agent3.pm
CONNECT CHILD backus:CoreDelete:1.02:00 - 989563418 CoreDelete Delete all Core f
iles in differents directories
STATUS_UPDATE NOTYPE backus:CoreDelete:1.02:00 - 989563423 Status => 0, Message
=> '0'
stop
DISCONNECT CHILD backus:CoreDelete:1.02:00 - 989563426 STOP command or message
bash-2.03$
```

CHAPITRE V CODES ET COMMENTAIRES

5.1. Agent SU.

5.1.1. Code

```
#!/usr/local/bin/perl -w
#
# ============================================================================
# This agent has been developped at: Institut d'Informatique des Facultés
# Universitaires Notre-Dame de la Paix à Namur
#
# ============================================================================
#
# This agent check for bad SU command attempts.
# The SU agent use the SULOG file contained in /var/adm/ directory
#
# Integrated in AAFID project developped by: COAST Laboratory, 1998.
# AAFID's author: Diego ZAMBONI and Eugene SPAFFORD
#
# Agent author:Simon Alexandre, Apr 15, 2001.
#
#
#
# NOTE: This file is in Perl's POD format. For more information, see the
#       manual page for perlpod(1).
#

package SULOG;

# The following makes the agent version number keep up with the RCS
# version number. The self-assignment at the end keeps the -w switch
# from complaining (because $VERSION may not be used here, but it is
# used in our base class).

$VERSION = do { my @r = (q$Revision: 1.2 $ =~ /\d+/g); sprintf "%d."."%02d" x $#r,
@r }; $VERSION = $VERSION;

##############################################################################
# AGENT PARAMETERS
#
%PARAMETERS=(
             Description    => "Check for repeated SU attempts",
             # CheckPeriod: delay between two checks.
             CheckPeriod          => 300,  # Seconds
             # Directory packaging SULOG file
             MyFileToCheck        => '/var/adm/sulog',
              SumPeriod      => 300, # Seconds
```

```
            );
##############################################################################

use AAFID::Agent;
use AAFID::Common;
use Util::NumQueue;
use Mail::Mailer;

use vars qw(
            @ISA
             $VERSION
             %PARAMETERS
            );

# Define the superclasses of our agent.
@ISA=qw(AAFID::Agent);

=head1 Description
Report number of failed login messages in the last SumPeriod seconds.
=head1 Implementation
Here we go.
=cut

# Agent initialization
sub Init {
  my $self=checkref(shift);
  my $queue=Util::NumQueue->new;
  $self->openfile($self->getParameter('MyFileToCheck'));
  $queue->timelength($self->getParameter('SumPeriod'));
  $self->setParameter(MyQueue => $queue);
  return $self;
}

# Logfile SULOG opening
sub openfile {
  my $self=checkref(shift);
  my $fname=shift;
  if ($fname) {
    my $file=IO::File->new($fname);
    if (!$file) {
      $self->Log("errors", "Error opening $fname: $!\n");
      return undef;
    }
    $self->Log("debug", "File $fname opened successfully.\n");
    $self->setParameter(MyFileHandle => $file);
    $file->seek(0,2);
    return $self;
  }
```

```
  else {
```

```
    $self->Log("debug", "No file name provided, waiting.\n");
    return $self;
  }
}

# check loop
sub Check {
  my $self=checkref(shift);
  my $queue=$self->getParameter('MyQueue');
  my $file=$self->getParameter('MyFileHandle');
  my ($date,$hour,$min,$result,$port,$user,$newuser);
  my $message= "";

  if ($file) {
    my $count=$self->getParameter('MyCount') || 0;
    $file->seek(0,1);
    while (<$file>) {
      $self->Log("debug", "Line read: $_");

          #Given model is the following  ' - ' which mean failed SU command
    next unless /\s-\s/;
      chop;
      ($date,$hour,$min,$result,$port,$user)=
      (split(/[\s:]+/))[1,2,3,4,5,6];
      $message = $message ."User=> ". $user ." tried to become Root at date
      => ".$date ." Hour=> ". $hour.":".$min. " From=> ". $port ."\n" ;

      $count++;
      $queue->add(1);
      $self->setParameter('MyCount' => $count);
    }
    $count=10 if $count>10;
    $self->Log("debug",
                  "Returning ($count, Failed SU attempts: $count)\n");
    my $status=$queue->sum;

    #send a mail to system administrator

    my $from = "SUagent\@info.fundp.ac.be";
    my $to   = "root\@info.fundp.ac.be";
    my $body = " Agent SU reached status 5 ! \n $message ";
    if ($status >= 5)
        {
        my $mailer = Mail::Mailer->new();
        $mailer->open ({ From => $from,
                         To   => $to,
                               Subject => "Agent SU Alert !!",
                       })
                or die "Can't open: $! \n";
        print $mailer $body;
        $mailer->close();
        }
```

```
    return ($status>10?10:$status, "$status");
  }
  return (0, "No file to monitor has been specified");
}

# Syntax: SETFILE File => 'file'
sub command_SETFILE {
  my ($self, $message, %params)=@_;
  if ($params{File}) {
    $self->Log("debug", "Setting file to monitor: $params{File}\n");
    $self->setParameter(MyFileToCheck => $params{File});
    if ($self->openfile($self->getParameter('MyFileToCheck'))) {
      return undef;
    }
    else {
      return {Error => "Error opening $params{File}: $!"};
    }
  }
  else {
    $self->Log("debug", "No File parameter provided.\n");
    return undef;
  }
}

_EndOfEntity;
```

5.1.2. Commentaires

Ainsi que nous l'avons expliqué dans les spécifications, l'agent va vérifier le fichier SULOG avec une fréquence que l'on va pouvoir paramétrer de manière dynamique. En effet, la fréquence de contrôle est représentée par le paramètre *CheckPeriod* qui peut être, à tout moment de l'exécution de l'agent, modifiée par l'utilisateur. Le second paramètre essentiel de cet agent est bien entendu le paramètre *MyFileToCheck* qui contient le chemin du fichier à vérifier.

L'agent, au moment de son exécution, va essayer d'ouvrir le fichier contenu dans le paramètre *MyFileToCheck* et, en cas de réussite, il va commencer la vérification en passant dans la boucle *Check*.

Le principe de cette boucle est simple. Les nouvelles lignes du fichier qui a été ouvert, vont être lues afin de vérifier qu'elles ne contiennent pas le modèle que l'on a défini :

```
Next unless /\s-\s/;
```

Si ce modèle signe d'une tentative SU échouée (' – ', espace tiret espace) est rencontré les instructions suivantes sont exécutées. On extrait alors les divers éléments de la ligne suspecte

en les isolants dans les diverses variables déclarées à cet effet (`$date`, `$hour`, `$min`, `$result`, `$port`, `$user`). Les divers éléments importants sont alors intégrés dans un message. La tentative détectée est comptabilisée et l'agent recommence sa vérification du fichier.

Si l'agent atteint un statut (`$status`) qui est supérieur ou égal à cinq, le contenu de la variable `$message` est alors intégrée dans un e-mail envoyé à l'administrateur système afin de l'avertir de la situation.

Enfin, la sortie standard de l'agent est respectée. A chaque itération le statut et le message qui contient le nombre de tentatives comptabilisées sont retourné à la console ou au contrôleur de l'agent.

5.2. Agent DiskSpaceCheck

5.2.1. Code

```
#!/usr/local/bin/perl -w
#
# ============================================================================
# This agent has been developped at: Institut d'Informatique des Facultés
# Universitaires Notre-Dame de la Paix à Namur
#
# ============================================================================
#
# This agent check that one or more disk partition doesn't exceed the limit
# authorized by the system administrator
#
# Integrated in AAFID project developped by: COAST Laboratory, 1998.
# AAFID's author: Diego ZAMBONI and Eugene SPAFFORD
#
# Agent author:Simon Alexandre, Apr 15, 2001.
#
#
# NOTE: This file is in Perl's POD format. For more information, see the
#       manual page for perlpod(1).
#

package DiskSpaceCheck;

# The following makes the agent version number keep up with the RCS
# version number. The self-assignment at the end keeps the -w switch
# from complaining (because $VERSION may not be used here, but it is
# used in our base class).
$VERSION = do { my @r = (q$Revision: 1.2 $ =~ /\d+/g); sprintf "%d."."%02d" x $#r,
@r }; $VERSION = $VERSION;

##############################################################################
# AGENT PARAMETERS
#

%PARAMETERS=(
             Description    => "Check for Disk Space exceeding",
             # CheckPeriod: delay between two checks.
             CheckPeriod         => 43200,  # Seconds
             # Directory to check
             DirectoryToCheck  => '/home/rouge/students/',
             SumPeriod          => 300, # Seconds
             LimitSize          => 100000, #100Mb
             );
##############################################################################
use AAFID::Agent;
```

```
use AAFID::Common;
use Util::NumQueue;
use Mail::Mailer;

use vars qw(
            @ISA
            $VERSION
            %PARAMETERS
            );

# Define the superclasses of our agent.
@ISA=qw(AAFID::Agent);

=head1 Description
Report number of failed login messages in the last SumPeriod seconds.
=head1 Implementation
Here we go.
=cut

# Agent initialization
sub Init {
  my $self=checkref(shift);
  my $queue=Util::NumQueue->new;
  $queue->timelength($self->getParameter('SumPeriod'));
  $self->setParameter(MyQueue => $queue);
  return $self;
}

# check loop
sub Check {
  my $self=checkref(shift);
  my $queue=$self->getParameter('MyQueue');
  my $count=$self->getParameter('MyCount') || 0;
  my $directory=$self->getParameter('DirectoryToCheck');
  my $limite=$self->getParameter('LimitSize');
  my $status=0;
  my $message = "USERS WITH AT LEAST 10*LIMIT OF DISK SPACE   ";
        my $mail = "root@info.fundp.ac.be;"; #contain user to prevent by mail
  #mail message that will be sent
  my $mail_message = "LIST OF BAD USERS  \n ";


  #change the current directory
  chdir ($directory) || die ("Invalid directory");

  # this command provide the disk space occuped by each user
  my @lines = `du -sk * | sort -rn`;

  # Extract names and disk sizes
  my @espaces = map {(split)[0]}@lines;
  my @noms = map {(split)[1]}@lines;
  my $taille =@noms;
```

```
# Check  each disk space
my $count_loop=0; # loop counter
while ($count_loop <= $taille)
  {
  my $temp=0;
  if ($espaces[$count_loop])
     {
     $temp=0+$espaces[$count_loop];
     }

  if ($temp > $limite )
     {
     # user added to the mail
     my $login= $noms[$count_loop];
     $mail = $mail .$login."\@info.fundp.ac.be".";";
     # Mail body = LOGIN + taille
     my $esp = 0 + $espaces[$count_loop];
          $mail_message = $mail_message . $login ."  Espace Disque= ".
     $esp .";\n";

     # $status update
     # 1er CASE: between limit and 2* authorized limit
     my $limite_min = $limite;
     my $limite_max = $limite*2;

                  if (($espaces[$count_loop] > $limite_min)
                 && ($espaces[$count_loop] < $limite_max)  )
               {
               # add 0.5 in status
               $count=$count+0.5;
               $queue->add(0.5);
               $self->setParameter('MyCount' => $count);
               }

     # 2eme CASE: between *2 limit and 4* authorized limit
     $limite_min = $limite *2;
     $limite_max = $limite *4;
                  if (($espaces[$count_loop] >= $limite_min)
               && ($espaces[$count_loop] < $limite_max))
               {
               # add 1 in status
               $count=$count+1;
               $queue->add(1);
                    $self->setParameter('MyCount' => $count);
               }
```

```
# 3eme CASE: between 4* limit and 10* authorized limit
      $limite_min = $limite *4;
      $limite_max = $limite *10;
                    if (($espaces[$count_loop] >= $limite_min)
                 && ($espaces[$count_loop] < $limite_max))
                 {
                 # add 2 in status
                 $count=$count+2;
                 $queue->add(2);
                 $self->setParameter('MyCount' => $count);
                 }

       # 4eme CASE: more than 10* authorized limit
       $limite_max = $limite *10;
       if (($espaces[$count_loop] >= $limite_max))
            {
            # add 3 in status
            $count=$count+3;
            $queue->add(3);
            $self->setParameter('MyCount' => $count);
            $message = $message . $login . ";\n";
            }
     }
   $count_loop++;
   }
   $count=10 if $count>10;
   $self->Log("debug","Returning ($count, total status: $count)\n");
   $status=$queue->sum;
   $queue->clear;

   #send a mail to system administrator

   my $from = "DiskSpaceCheckAgent\@info.fundp.ac.be";
   my $to = $mail;
   my $body = " Agent DiskSpace reached status 10 !\n $mail_message \n
                $message";
   if ($status >= 10)
       {
       my $mailer = Mail::Mailer->new();
       $mailer->open ({ From => $from,
                        To   => $to,
                             Subject => "Agent DiskSpace Alert !!",
                      })
              or die "Can't open: $! \n";
       print $mailer $body;
       $mailer->close();
       }

   return ($status>10?10:$status, "$message");
}

_EndOfEntity;
```

5.2.2. Commentaires

L'agent DiskSpaceCheck va vérifier l'espace disque occupé par un ensemble d'utilisateurs et générer une alerte si la limite autorisée est dépassée. Le principe suivi pour implémenter cet agent est relativement simple. Sur base du répertoire passé en arguments (*DirectoryToCheck*), l'agent va, dans sa boucle de contrôle, se positionner sur ce répertoire (`chdir ($directory)`).

Dans ce répertoire, la commande *du* est employée afin de passer en revue tous les sous répertoires de *DirectoryToCheck* qui correspondent aux partitions des utilisateurs. Le résultat de cette commande est redirigé dans un tableau.

```
my @lines = `du -sk * | sort -rn`;
```

Cette manière de procéder présente l'avantage de disposer d'un tableau composé des éléments suivants « 26541 user » c'est-à-dire l'espace et le nom du répertoire. En outre, tous ces éléments sont triés sur la taille de manière décroissante. Les deux lignes suivantes permettent d'extraire les deux éléments afin de les ranger dans deux tableaux différents (`@espaces` et `@noms`).

```
my @espaces = map {(split)[0]}@lines;
my @noms = map {(split)[1]}@lines;
```

La première ligne sépare le premier élément (`[0]`) séparé du second par un espace blanc (le modèle par défaut de la fonction *split*) et le range dans `@espaces` . La seconde ligne fonctionne de manière identique. Après leur exécution, nous disposons donc de deux tableaux triés contenant les noms des utilisateurs et l'espace disque qu'ils occupent respectivement.

Sur cette base, l'agent va maintenant pouvoir vérifier si chacun des utilisateurs respecte ou non la limite imposée. Pour ce faire, nous avons précédemment distingué quatre cas de figure allant du simple dépassement sans gravité à l'utilisation abusive d'espace disque. Selon qu'un utilisateur se trouve dans une des quatre situations, le statut est diversement incrémenté.

En outre, étant donné que nous avons prévu d'avertir, d'une part l'administrateur de la situation détectée par l'agent et d'autre part les utilisateurs en défaut, nous devons récupérer les données au fur et à mesure. Il s'agit de prendre les logins des utilisateurs à avertir. Pour ce faire, nous prenons le nom contenu dans `@noms` qui correspond au login de l'utilisateur et nous l'ajoutons à la variable : `my $login= $noms[$count_loop];`

```
$mail = $mail .$login."\@info.fundp.ac.be".";";
```

Le tout est incorporé à `$mail` qui sera utilisée comme adresse de destination du mail envoyé. De plus, le corps du message (`$mail_message`) qui sera envoyé est composé petit à petit par les informations collectées soit le login et l'espace disque occupé.

L'avant dernière opération exécutée par l'agent est l'envoi de ce mail à l'administrateur système et aux utilisateurs concernés. Enfin ,l'agent peut retourner son statut.

5.3. Agent FtpCheck

5.3.1. Code

```
#!/usr/local/bin/perl -w
#
# ============================================================================
# This agent has been developped at: Institut d'Informatique des Facultés
# Universitaires Notre-Dame de la Paix à Namur
#
# ============================================================================
#
# This agent check for suspect FTP deposit
# The agent use the xferlog log file contained in /var/adm/ directory
# Only deposit from the outside are checked. If the  remote host
# adress contain either 'fundp' or '138.48.' which indicate the intern
# origin of the deposit.
#
# Integrated in AAFID project developped by: COAST Laboratory, 1998.
# AAFID's author: Diego ZAMBONI and Eugene SPAFFORD
#
# Agent author:Simon Alexandre, Apr 15, 2001.
#
#
# NOTE: This file is in Perl's POD format. For more information, see the
#       manual page for perlpod(1).
#

package FtpCheck;

# The following makes the agent version number keep up with the RCS
# version number. The self-assignment at the end keeps the -w switch
# from complaining (because $VERSION may not be used here, but it is
# used in our base class).

$VERSION = do { my @r = (q$Revision: 1.2 $ =~ /\d+/g); sprintf "%d."."%02d" x $#r,
@r }; $VERSION = $VERSION;

##############################################################################
# AGENT PARAMETERS
#
%PARAMETERS=(
            Description     => "Check FTP suspect incoming",
            # CheckPeriod: delay between two checks.
            CheckPeriod            => 1800,  # Seconds
            # Directory packaging XFERLOG file
            MyFileToCheck       => '/var/adm/xferlog',
            SumPeriod       => 2000, # Seconds
            LimitSize       => 500000,
           );
```

```
##############################################################################

use AAFID::Agent;
use AAFID::Common;
use Util::NumQueue;
use Mail::Mailer;

use vars qw(
            @ISA
            $VERSION
            %PARAMETERS
            );

# Define the superclasses of our agent.
@ISA=qw(AAFID::Agent);
=head1 Description
Report number of failed login messages in the last SumPeriod seconds.
=head1 Implementation
Here we go.
=cut

# Agent initialization
sub Init {
  my $self=checkref(shift);
  my $queue=Util::NumQueue->new;
  $self->openfile($self->getParameter('MyFileToCheck'));
  $queue->timelength($self->getParameter('SumPeriod'));
  $self->setParameter(MyQueue => $queue);
  return $self;
}

# Logfile XFERLOG opening
sub openfile {
  my $self=checkref(shift);
  my $fname=shift;
  if ($fname) {
    my $file=IO::File->new($fname);
    if (!$file) {
      $self->Log("errors", "Error opening $fname: $!\n");
      return undef;
    }
    $self->Log("debug", "File $fname opened successfully.\n");
    $self->setParameter(MyFileHandle => $file);
    $file->seek(0,2);
    return $self;
  }
  else {
    $self->Log("debug", "No file name provided, waiting.\n");
    return $self;
  }
}
```

```
# check loop
sub Check {
  my $self=checkref(shift);
  my $queue=$self->getParameter('MyQueue');
  my $file=$self->getParameter('MyFileHandle');

  #variable used to check the file size (50 Mb max)
  my $limit_size =$self->getParameter('LimitSize');

  #elements from the logs
  my ($day_name,$month, $day,$hour,$min,$year);
  my ($remote_host,$size,$filename);

  #message returned
  my $message = " ";
  my $pattern ="i";
  my $intern_adress;

  if ($file)
   {
    my $count=$self->getParameter('MyCount') || 0;
    $file->seek(0,1);
    while (<$file>)
    {
      $self->Log("debug", "Line read: $_");
      next unless /\si\s/;
      chop;
      ($day_name,$month, $day,$hour,$min,$year,$remote_host,$size,$filename)=
      (split(/[\s:]+/))[0,1,2,3,4,6,8,9,10];

      #check if the remote host came from the fundp
      $intern_adress = $remote_host !~ /fundp|138.48/;
      if ($intern_adress != 0)
         {
         if ($size >= $limit_size)
            {
             $message= $message ." Remote host => ".$remote_host. "  File size
            =>" .$size."\n" ;
            $count++;
            $queue->add(1);
            $self->setParameter('MyCount' => $count);
            }
        }
    }
    $count=10 if $count>10;
    $self->Log("debug","Returning ($count, Failed login messages:
    $count)\n");
    my $status=$queue->sum;
    #send a mail to system administrator

    my $from = "FtpCheckAgent\@info.fundp.ac.be";
    my $to = "root\@info.fundp.ac.be";
```

```
    my $body = " Agent FtpCheck reached status 5 !\n $message ";
    if ($status >= 5)
        {
        my $mailer = Mail::Mailer->new();
        $mailer->open ({ From => $from,
                         To   => $to,
                               Subject => "Agent FtpCheck Alert !!",
                       })
                or die "Can't open: $! \n";
        print $mailer $body;
        $mailer->close();
        }
    return ($status>10?10:$status, "$status");
  }
  return (0, "No file to monitor has been specified");
}


# Syntax: SETFILE File => 'file'
sub command_SETFILE {
  my ($self, $message, %params)=@_;
  if ($params{File}) {
    $self->Log("debug", "Setting file to monitor: $params{File}\n");
    $self->setParameter(MyFileToCheck => $params{File});
    if ($self->openfile($self->getParameter('MyFileToCheck'))) {
      return undef;
    }
    else {
      return {Error => "Error opening $params{File}: $!"};
    }
  }
  else {
    $self->Log("debug", "No File parameter provided.\n");
    return undef;
  }
}

_EndOfEntity;
```

5.3.2. Commentaires

L'agent FtpCheck va analyser le fichier *var/adm/xferlog* . Pour ce faire l'agent va, avant d'entrer dans la boucle de vérification, ouvrir le fichier *xferlog* . Dans la boucle de vérification, l'agent va effectuer l'instruction suivante `$file->seek(0,1);` Cette dernière lui permet de se placer à la position courante dans le fichier avant de ne pas prendre en compte toutes les lignes du fichier. [TIL 00 :397] Seules les nouvelles lignes seront donc analysées par l'agent.

Ces lignes vont être confrontées à un modèle très simple que nous avons défini et qui signifie qu'un fichier vient d'être déposé : `next unless /\si\s/;`

Si cette occurrence (espace i espace) est rencontrée, alors les divers éléments de la ligne sont extraits et rangés dans les variables définies à cet effet. `($day_name,$month, $day,$hour,$min,$year,$remote_host,$size,$filename)`. Ensuite, grâce à la variable `$size` nous allons vérifier s'il s'agit d'un dépôt important ou non. Si la taille du fichier déposé dépasse la taille limite définie, le statut de l'agent est incrémenté. En outre, afin d'avertir l'administrateur de cet état de fait, les éléments importants caractérisant ce dépôt (nom de l'hôte, taille du fichier) sont intégrés dans la variable `$message` qui servira de corps de message au mail envoyé à l'administrateur.

Enfin, avant de retourner son statut, l'agent envoie le mail à l'administrateur si le statut atteint ou dépasse une limite fixée à deux. Le choix de cette limite est purement théorique. Seule la pratique à moyen terme permettra de fixer une limite raisonnable afin de ne pas laisser passer des cas litigieux et de ne pas saturer inutilement d'e-mails l'administrateur.

5.4. Agent CoreDelete

5.4.1. Code

```
#!/usr/local/bin/perl -w
#
# ============================================================================
# This agent has been developped at: Institut d'Informatique des Facultés
# Universitaires Notre-Dame de la Paix à Namur
#
# ============================================================================
#
# Agent CoreDelete help the system administrator to clean directories.
# This agent will be used to clean core files from students directories
#
#
# Integrated in AAFID project developped by: COAST Laboratory, 1998.
# AAFID's author: Diego ZAMBONI and Eugene SPAFFORD
#
# Agent author:Simon Alexandre, Apr 15, 2001.
#
#
#
# NOTE: This file is in Perl's POD format. For more information, see the
#       manual page for perlpod(1).
#

package CoreDelete;

# The following makes the agent version number keep up with the RCS
# version number. The self-assignment at the end keeps the -w switch
# from complaining (because $VERSION may not be used here, but it is
# used in our base class).

$VERSION = do { my @r = (q$Revision: 1.2 $ =~ /\d+/g); sprintf "%d."."%02d" x $#r,
@r }; $VERSION = $VERSION;

##############################################################################
# AGENT PARAMETERS
#

%PARAMETERS=(
            Description => "Delete all Core files in differents directories",
            # CheckPeriod: delay between two checks.
            CheckPeriod              => 43200,# Seconds
            SumPeriod          => 300, # Seconds
            DirectoryToClean   => '/home/rouge/students/',
             );

##############################################################################
```

```
use AAFID::Agent;
use AAFID::Common;
use Util::NumQueue;
use Mail::Mailer;

use vars qw(
            @ISA
            $VERSION
            %PARAMETERS
            );
# Define the superclasses of our agent.
@ISA=qw(AAFID::Agent);
=head1 Description
Report number of failed login messages in the last SumPeriod seconds.
=head1 Implementation
Here we go.
=cut

# agent initialization
sub Init {
  my $self=checkref(shift);
  my $queue=Util::NumQueue->new;
  $queue->timelength($self->getParameter('SumPeriod'));
  $self->setParameter(MyQueue => $queue);
  return $self;
}

# check loop
sub Check {
  my $self=checkref(shift);
  my $queue= $self->getParameter('MyQueue');
  my $count= 0;
  my $status;
  my $directory = $self->getParameter('DirectoryToClean');

  # Change the working directory
  chdir ($directory) || die ("Invalid directory");

  # Command used to find core files and remove it
  my @arguments= ("find", ".", "-name", "core", "-exec", "rm", "{}", "\;");
  system (@arguments);

  $count=0;
  $queue->add(0);
  $self->setParameter('MyCount' => $count);
  $count=10 if $count>10;
  $status=$queue->sum;

  #send a mail to system administrator

  my $from = "CoreDeleteAgent\@info.fundp.ac.be";
```

```
  my $to = "root\@info.fundp.ac.be";
  my $body = " AAFID=>Agent CoreDelete Report  \n";
  my $mailer = Mail::Mailer->new();
  $mailer->open ({ From => $from,
                   To   => $to,
                         Subject => "Agent CoreDelete Work Done !!",
                 })
                 or die "Can't open: $! \n";
           print $mailer $body;
           $mailer->close();

  return ($status>10?10:$status, "$status");

}

_EndOfEntity;
```

5.4.2. Commentaires

Dernier agent implémenté, l'agent CoreDelete se distingue des autres agents par son orientation complète vers l'aspect administration du système. Son mode de fonctionnement et son implémentation sont donc très simples. Il va simplement effectuer sa tâche et générer un statut non incrémenté, donc égal à zéro.

Sur base du répertoire passé en argument (`DirectoryToClean`), l'agent va se positionner sur ce répertoire et effacer tous les fichiers Core qu'il va rencontrer. Les instructions suivantes effectuent cette tâche :

```
chdir ($directory) || die ("Invalid directory");
my @arguments= ("find", ".", "-name", "core", "-exec", "rm", "{}", "\;");
system (@arguments);
```

Le principe est simple. La commande *find* avec l'argument –name permet de rechercher tous les fichiers nommés core et ensuite grâce au paramètre *–exec* tous les fichiers trouvés sont effacés.

Lorsque l'agent a terminé, il envoie un mail à l'administrateur système afin de l'avertir qu'il a effectué sa tâche.

CONCLUSIONS ET PISTES DE RECHERCHE

Notre étude de l'administration d'un système tournant sous UNIX/SOLARIS nous a révélé l'ampleur et la difficulté de la tâche. C'est le poids de cette dernière que nous avons voulu atténuer en mettant au point les quatre agents présentés ci-dessus. Certes, comme cela a été souligné lors de la présentation de la sécurité des systèmes, cet aspect du travail d'un administrateur est à remettre sans cesse en question. Comme nous l'avons vu, les risques proviennent aussi bien de l'intérieur que de l'extérieur du système et toutes les brèches sont difficilement maîtrisables.

Cependant, en intégrant AAFID2 au système de l'Institut d'informatique, nous avons voulu respecter le double aspect de ce mémoire : administration et protection du système. En effet, AAFID2 est un IDS et donc entièrement orienté sécurité. Or nous avons décidé d'y intégrer des tâches d'administration qui, nous l'avons vu, peuvent revêtir les deux aspects. L'automatisation de ces tâches rendue possible par le mode de fonctionnement d'AAFID2 constitue un progrès considérable. Il faut cependant souligner les limites du système pour éviter toute future mauvaise surprise.

En effet, si l'on se limite à une utilisation simple d'AAFID2, ce qui signifie en d'autres termes, le lancement indépendant des divers agents pour exercer la surveillance de certaines parties du système, nous pouvons être certain de bénéficier pour ces points là uniquement d'une aide précieuse. Toutefois, une utilisation d'AAFID2 dans son ensemble, donc en lançant des moniteurs et des contrôleurs chargés de gérer les agents et les alarmes qu'ils provoquent reste sujette à caution. Les auteurs d'AAFID2 le signalent dans la documentation et nous l'ont récemment rappelé. Le comportement des agents est garanti, excepté les erreurs de programmation bien entendu. Mais l'utilisation de l'ensemble du système présente encore des bugs sur lesquels ils travaillent. Ils exhortent donc à utiliser AAFID2 de manière ponctuelle. Par conséquent, l'utilisation qui sera faite d'AAFID2 à l'institut restera limitée aux agents utiles à l'administration du système.

AAFID2 en tant qu'IDS est susceptible en cas d'attaque importante d'être la première cible des pirates. En effet, dans la plupart des cas, les pirates cherchent à repérer l'IDS pour le mettre hors de combat et pouvoir intervenir à leur guise sur le système. AAFID2 n'échappe donc pas à cette règle. Notre IDS étant implémenté en Perl5, lorsqu'un ou plusieurs agents tournent sur le serveur, ils sont simplement référencés par l'intitulé *perl* et non pas leur nom. On ne peut donc pas dissimuler leur présence aux yeux curieux d'un éventuel intrus.

L'inconvénient majeur des IDS, souligné par bon nombre de spécialistes, reste la rapidité de répercussion de l'alerte détectée. Effectivement, lorsque l'agent remarque un fait exceptionnel et le signale il faut que l'administrateur système ou un autre responsable soient devant leur console pour voir cette alerte. Cet état de fait nous a donc conduit, avec l'accord des auteurs d'AAFID2, à introduire en plus l'envoi, par l'agent, d'un e-mail à l'administrateur afin de l'avertir de la situation.

Bien entendu, pour que l'alerte soit repérée, il faut que cet e-mail soit lu rapidement ce qui n'est pas sûr. C'est pourquoi, nous pensons à l'intégration possible d'un système complémentaire à nos agents et à AAFID. Il s'agit d'avertir l'administrateur dès que l'alerte est déclenchée, ce qui se traduit par l'envoi du mail. Pour diminuer le délai d'avertissement, il faudrait que l'administrateur soit prévenu par téléphone de la situation.

Grâce à la combinaison possible de la technologie e-mail et des SMS on pourrait réduire cette limitation du système. Cette technologie fonctionne sur le principe de notification. Concrètement, dès qu'un e-mail est reçu, une notification de l'arrivée du nouveau message est faite via SMS sur le téléphone portable du responsable. A l'heure actuelle, un opérateur de téléphonie mobile belge propose ce service. Une formule, payante, est susceptible de nous intéresser. Elle permet tout d'abord de préciser le niveau de priorité de l'e-mail et la plage horaire durant laquelle on pourra recevoir ces notifications. Ensuite, chaque notification peut comporter un certain nombre d'éléments de base comme l'auteur ou surtout le sujet. Bien entendu, deux conditions préalables : être client de cet opérateur et créer une adresse e-mail chez lui.

Toutefois, au-delà des considérations commerciales sous-jacentes, nous pouvons voir tout l'intérêt d'un tel système. Il nous permettrait de pallier le problème de lenteur et de disposer d'un IDS avec détection quasi en temps réel.

Compte tenu de cette possibilité, on peut envisager le déploiement complet d'AAFID et de ses agents sur les machines de l'institut. Pour cela, il faut dans un premier temps intégrer dans tous les agents utilisés l'envoi d'un e-mail en cas d'alerte. En second lieu, le test minutieux des divers agents s'impose car ils ont été développés sur des machines parfois diversement configurées. Les utiliser sans vérification conduirait à l'introduction de nouvelles brèches dans la sécurité du système que l'on pourrait croire bien protéger alors qu'il n'en est rien. L'objectif final de cette démarche serait de disposer d'un IDS fonctionnel capable de générer des messages relayés grâce aux SMS et transmis presque en temps réel à l'administrateur système. Enfin, de nouveaux agents pourraient être implémentés, en suivant la même logique, afin de protéger d'autres aspects du système.

La démarche que nous avons poursuivie tout au long de ce travail nous a permis de combiner deux volets importants de la gestion d'un système, c'est-à-dire l'administration et la sécurité et de les intégrer à un projet existant afin d'augmenter la sécurité souvent mise à l'épreuve de nos systèmes.

BIBLIOGRAPHIE

1. Documents consultés pour information

1.1. Linux, Unix, Solaris

ARMSPACH, J.-P., COLIN, P., OSTRE-WAERZEGGERS, F., *LINUX. Initiation et utilisation*, Paris, 2000.

ARMSPACH, J.-P., COLIN, P., OSTRE-WAERZEGGERS, F., *Unix. Initation et utilisation*, Paris, 1999.

DREGGER, E., GUTMANN, M., LANNERT et a., *Linux. Installation – Internet Administration*, (éd. Micro Application), Paris, 2000.

FRISCH, A., *Essential system Administration*, (éd. O'Reilly), Sebastopol, 1995.

TACKETT, J., BURNETT, S., *Linux*, (éd. Le MacMillan. Campus Press), Paris, 2000.

http://www.commentcamarche.net

1.2. Sécurité informatique

AMOROSO, E., *Intrusion detection: an introduction to Internet surveillance, correlation, trace back, traps, and response*, New-York, 1999.

DENNING, P., *Computers under attack : intruders, worms and viruses*, (éd. ACM Press), 1990.

STALLINGS, W., *Network and Internetworking Security : Principles and Practice* , (éd. Prenctice Hall), 1995.

KAEO, M., *Sécurité des réseaux*, Paris, 2000.

PELISSIER, Ch., *Guide de sécurité des systèmes UNIX*, Paris, 1993.

http://www.commentcamarche.net

http://psyon.terrashare.com/index.html

1.3. Perl

SCHWARTZ, R., CHRISTIANSEN, T., *Learning Perl*, (éd. O'Reilly), 1997.

TILL, D., *Perl 5*, (éd. Campus Press), Paris, 2000.

VROMANS, J., *Perl 5 Pocket Reference*, (éd. O'Reilly), 2000.

http://www.perldoc.com

http://www.perl.com

2. Documents utilisés dans le mémoire

1.1. Linux, Unix, Solaris

[NEM 95] NEMETH, E., *Unix system administration handbook*, Londres, 1995.

[WIE 00] WIELSCH, M., *Grand Livre UNIX*, (éd. Micro Application) , Paris, 2000.

[PEL 96] PELISIER, Ch., *UNIX. Utilisation, Administration système et réseau*, Paris, 1996.

1.2. Sécurité informatique

[ALL 00-a] ALLEN, J., CHRISTIE, A., & a., *State of the Practice of Intrusion Detection Technologies*, Technical Report. CMU/SEI-99-TR-028, Pittsburgh, 2000. http://www.sei.cmu.edu/publications/documents/99.reports/99tr028/99tr028abstract.html

[ALL 00-b] ALLEN, J., *The IDS Life Cycle* dans *Infosec Outlook* (CERT), vol.1, n°5, 2000.

http://www.cert.org/infosec-outlook/infosec_1-5.html

[BRU 99] BRUMLEY, D., *Invisible intruders: rootkits in practice* dans *;login:, special issue. Intrusion Detection*, sept. 1999, p.27-29.

[CER 00] *CERIAS Security Visionary roundtable. Call to Action.* Accenture, 2001.

[CERT 00-a] www.cert.org\security-improvement\practices\p091.html

[CERT 00-b] www.cert.org\tech_tips\intruder_detection_checklist.html

[CERT 00-c] www.cert.org\tech_tips\security_tools.html

[DEB 99] DEBAR, H., DACIER, M. et WESPI, A., *Towards a taxonomy of intrusion-detection systems* dans *Computer Networks*, vol. 31, n°8, Amsterdam, 1999, p. 805-822.

[DIS 00] *Distributed Denial of Service attacks.* AusCERT, 2000.

http://www.auscert.org.au/Information/Auscert_info/whatis.html

[FRI 00] FRINCKE, D., HUANG, M., *Recent advances in intrusion detection systems* dans *Computer Networks*, vol. 34, n°4, Amsterdam, 2000, 541-544.

[GAR 96] GARFINKEL, S., SPAFFORD, G., *Practical UNIX and Internet Security. Second edition,* Sebastopol, 1996.

[HOW 98] HOWARD, J., LONGSTAFF, T., *A Common Language for Computer Security Incidents*, Sandia National Laboratories, Albuquerque-Livermore, 1998.

http://www.cert.org/research/taxonomy_988667.pdf

[HUA 99] HUANG, M, JASPER, R. et WICKS, T., *A large scale distributed intrusion detection framework based on attack strategy analysis* dans *Computer Networks*, vol. 31, n°23-24, Amsterdam, 1999, p. 2465-2475.

[HUG 00] Mc HUGH, J., *Defending Yourself: The Martial Arts of Intrusion Detection* dans *Infosec Outlook* (CERT), vol.1, n°4, 2000.

http://www.cert.org/infosec-outlook/infosec_1-4.pdf

[KOS 99] KOSSAKOWSKI, K.-P., ALLEN, J., et a., *Responding to Intrusions*, Pittsburgh, 1999.

http://www.sei.cmu.edu/publications/documents/doc.list/1999.htm

[KUM 95] KUMAR, S., *Classification and Detection of Computer Intrusions*, Purdue, 1995.

[MAN 00] MANGANARISI, S., CHRISTENSEN, M. et a., *A data mining analysis of RTID alarms* dans *Computer Networks*, vol. 34, n°4, Amsterdam, 2000, p. 571-577.

[MEA 00] MEAD, N., ELLISON, R.. et a., *Survivable Network Analysis Method*, Pittsburgh, 2000.

http://www.sei.cmu.edu/publications/documents/00.reports/00tr013.html

[NSA 99] *NSA Glossary of Terms in Security and Intrusion Detection*, SANS Institute Resources

http://www.sans.org/newlook/resources/glossary.htm

[PIC 94] PICHNARCZYK, K., WEEBER, S. et FEINGOLD, R., *Unix Incident Guide: How to Detect an Intrusion. CIAC-2305 R.1*, Livermore, 1994.

http://ciac.llnl.gov/cgi-bin/index/documents

[PUK 96] PUKETZA, N., ZHANG, H. et a., *A Methodology for Testing Intrusion Detection Systems*, Davis, 1996.

[RES 99] *Results of the Distributed-Systems Intruder Tools Workshop. Pittsburgh, Pennsylvania USA November 2-4, 1999* , CERT, Pittsburgh, 1999.

http://www.cert.org/reports/dsit_workshop-final.html

[SUN 96] SUNDARAM, A., *An Introduction to Intrusion Detection*, 1996.

http://www.acm.org/crossroads/xrds2-4/intrus.html

[TAB 98] TABER, M., *Maximum Security: A Hacker's G Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network*, Macmillan, 1998.

[WES 98] WEST-BROWN, M., STIKVOORT, D. et KOSSAKOWSKI, K.-P., *Handbook for Computer Security Incident Response Teams (CSIRTs)*, CERT, Pittsburgh, 1998.

http://www.sei.cmu.edu/pub/documents/98.reports/pdf/98hb001.pdf

1.3. AAFID

[AAF 99] Codes et Documentation d'AAFID2.

[BAL 98] BALASUBRAMANIYAN, J., SPAFFORD, E., ZAMBONI, D., et a., *An Architecture for Intrusion Detection using Autonomous Agents* dans *Technical Report 98/05*, COAST Laboratory, Department of Computer Sciences, Purdue, 1998.

[SPA 98] SPAFFORD, E. et ZAMBONI, D., *AAFID2 User Guide*, Purdue, 1998.

[SPA 00] SPAFFORD, E. et ZAMBONI, D., *Intrusion detection using autonomous agents* dans *Computer Networks*, vol. 34, n°4, Amsterdam, 2000, p. 547-570.

1.4. Presse

[SOIR 00-a] *Panne de trois heures aux Etats-Unis. Un raid déconnecte Yahoo, le géant de l'Internet* dans *Le Soir. Actualité économique*, 9/02/2000.

[SOIR 00-b] MATTHEIEM, N.,*Le FBI piste les cybervandales,* dans *Le Soir. Actualité économique*, 10/02/2000.